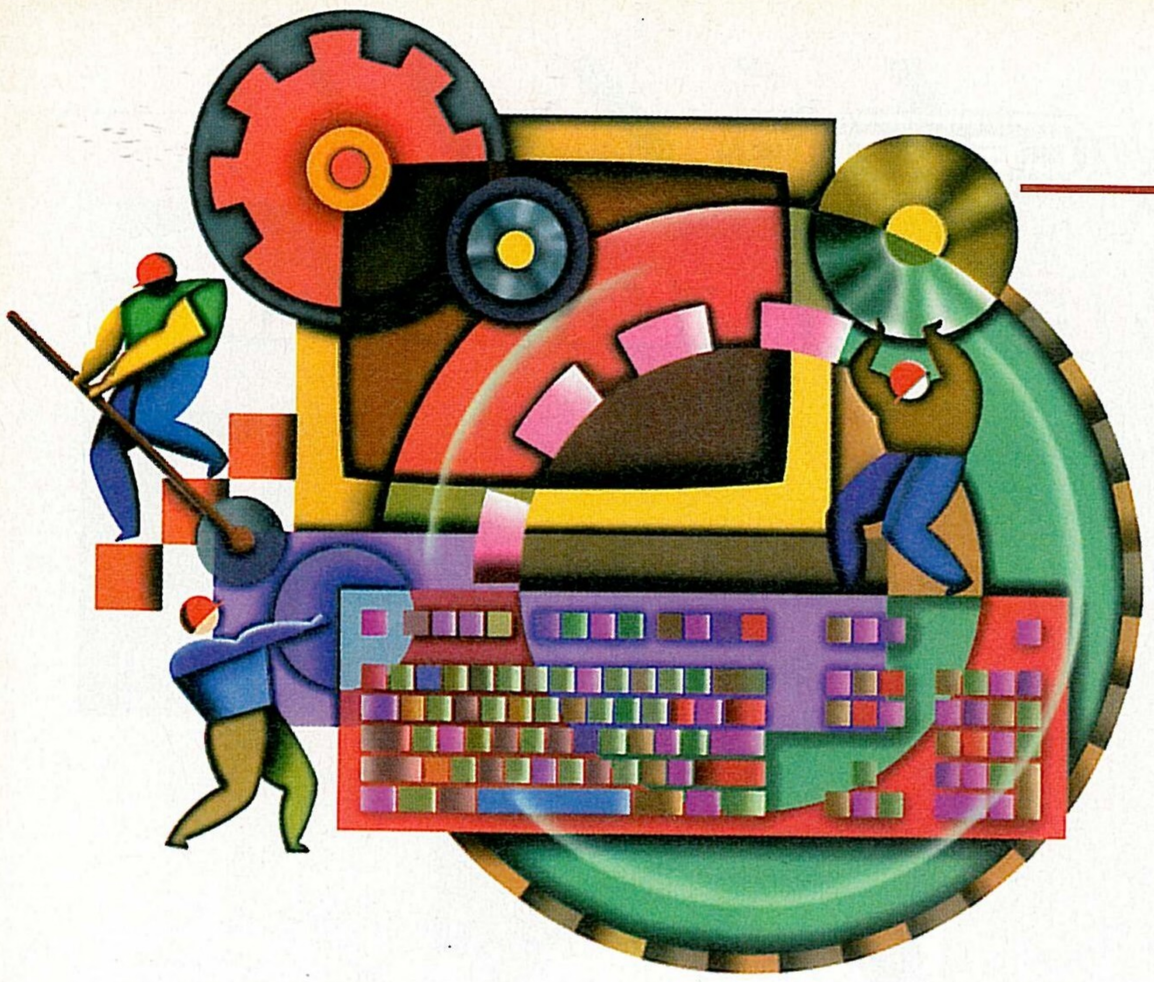
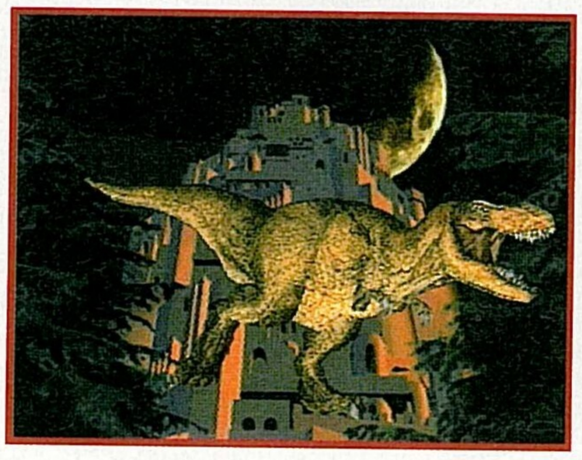


Programar videojuegos dejará de ser, desde hoy, un hobby para unos pocos privilegiados. Gracias al lanzamiento del kit de desarrollo «casero» **NET YAROZE**, **Sony** propone volver a los tiempos en que cualquiera podía hacerse famoso (y rico, por qué no) desde el ordenador de su casa. Este mes abrimos una nueva sección en la que trataremos de explicaros todos los secretos de la programación.

DE VUELTA A LA DECADA DE LOS 80

Si muchos de vosotros habéis sentido alguna vez el gusanillo de la programación, estáis de enhorabuena. Desde este mismo mes os proponemos una nueva forma de ver el mundo de los videojuegos, y todo ello gracias al kit de desarrollo «casero» que **Sony** a puesto a la venta, el ya famoso **NET YAROZE**. Os explicaremos desde el funcionamiento interno de ***PlayStation*** hasta las técnicas de programación más comunes, pasando por el funcionamiento de las herramientas que se nos entregan y, cómo no, el lenguaje de programación C, cuyo conocimiento resulta imprescindible en la aventura que nos proponemos a afrontar. Sabemos lo complicado que puede resultar todo esto (estamos seguros de que en un principio todo os sonará a chino), pero con un poco de paciencia y perseverancia, los resultados pueden ser sorprendentes. En esta nueva sección tendrá cabida todo aquello relacionado con el tema de la programación y el desarrollo de videojuegos, aunque siempre aplicado a las posibilidades que el *kit* **NET YAROZE** nos brinda. Intentaremos acercar las opiniones de los programadores, mostrar cómo se hizo este o aquel juego e, incluso, explicar los secretos mejor guardados de cada uno de ellos.

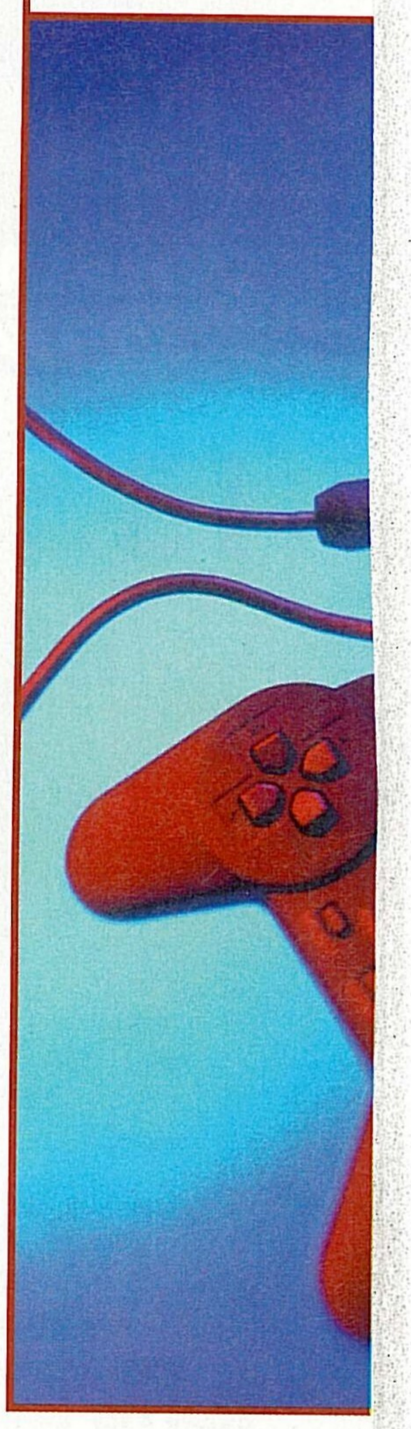


Aún así, lo mejor para empezar es conocer el entorno en el que nos moveremos, por lo que resultará conveniente echar una ojeada a los componentes de dicho *kit* de desarrollo. En este primer capítulo hablaremos de la página *WEB* de **NET YAROZE**, a la que se nos permitirá el acceso por medio de nuestro nombre y clave (que se nos entrega al adquirir el *kit*). También, y como es lógico, se hace un breve repaso sobre cada una de las herramientas de *software* que se incluyen, completando esta primera entrega con el sencillo programa «Hello World!», inevitable comienzo para todo curso de programación que se precie de serlo.

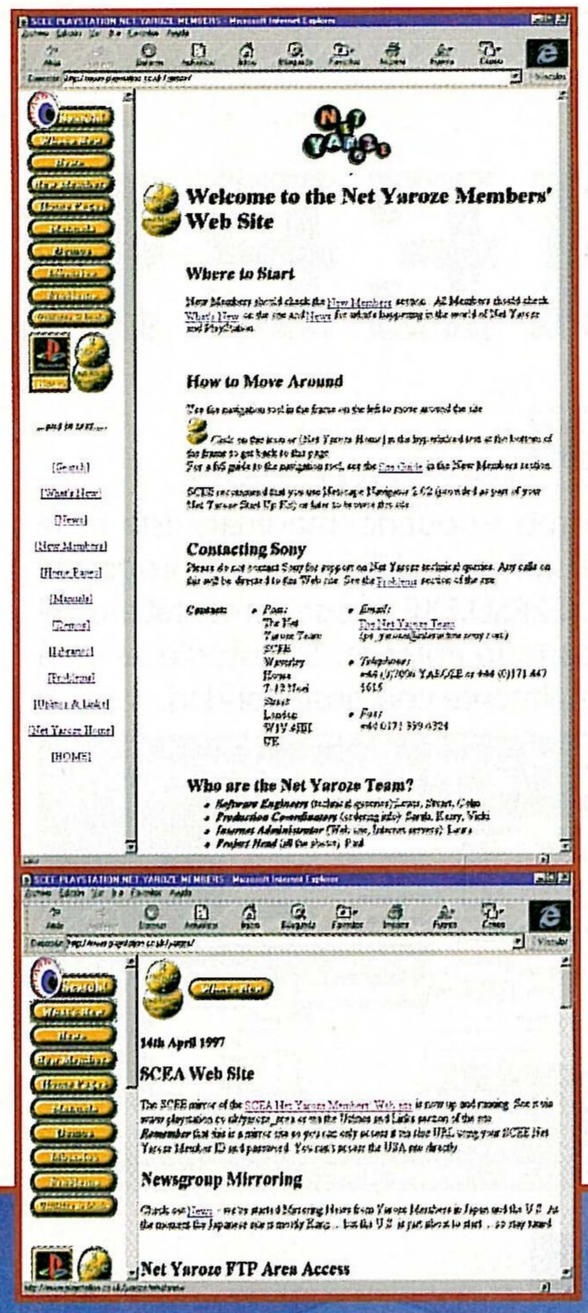
Si deseáis adquirir el *kit* **NET YAROZE** u obtener información sobre el mismo, sólo tenéis que llamar al número **007 44 (0) 171 447 1616**, escribir un E-Mail a **ps_yaroze@interactive.sony.com** o rellenar el formulario en **http://www.scee.sony.co.uk.**

El precio, ya sabéis, 550£ (al cambio unas 125.000 pesetas).

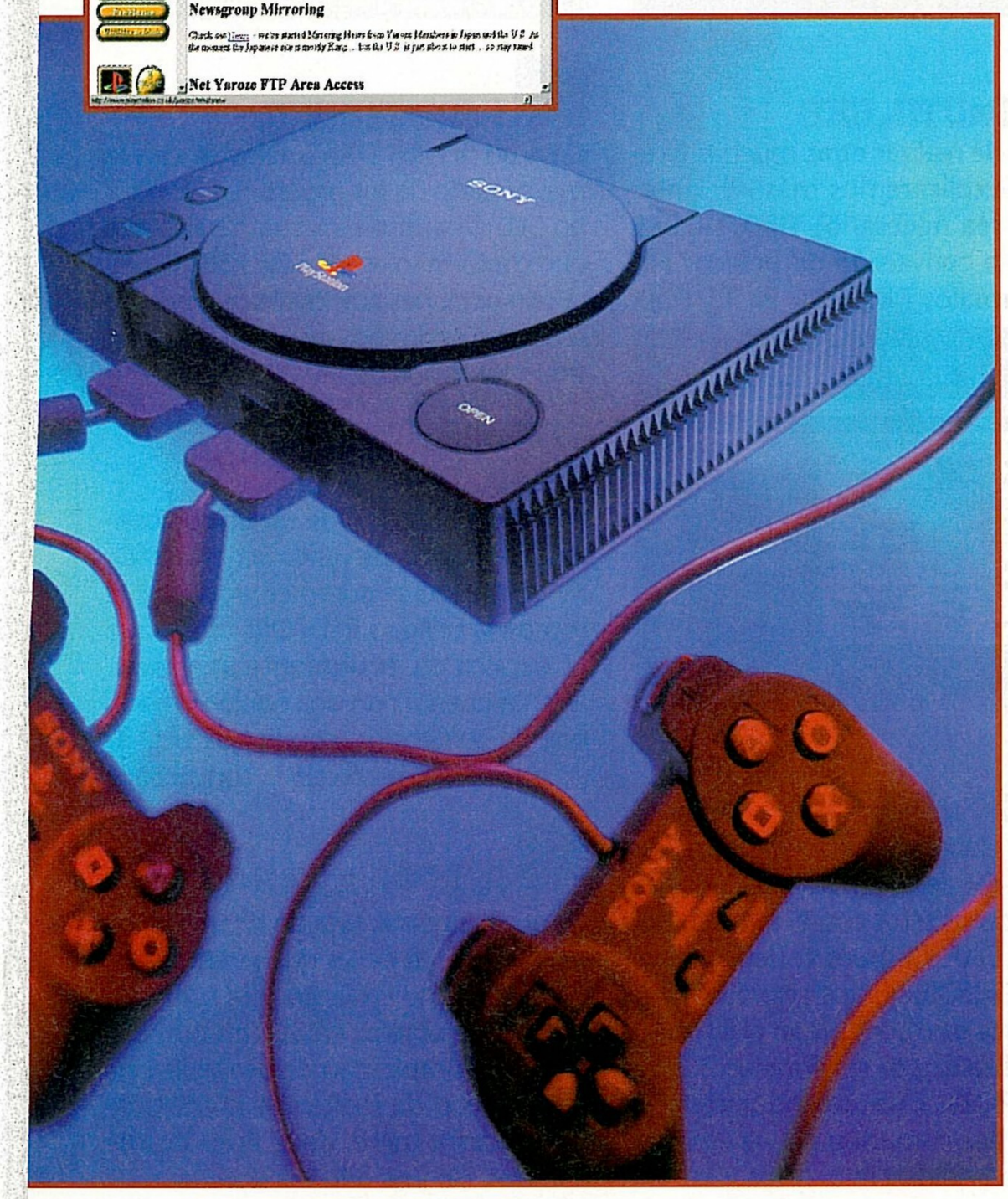
J. C. MAYERICK



aprende a PROGRAMAR con net yaroze



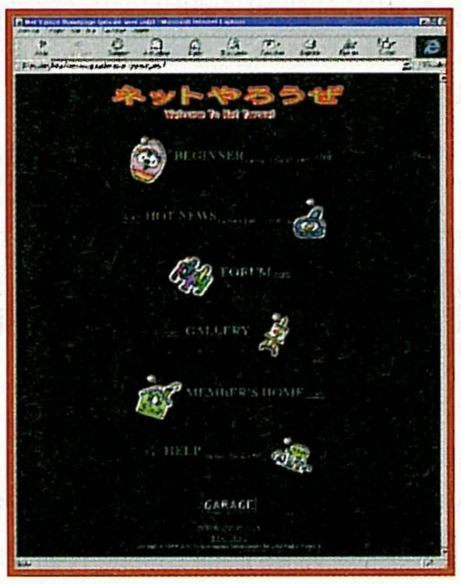
La página WEB es uno de los elementos a los que más importancia le ha dado **Sony**. Su idea es que a través de ella se distribuyan *demos*, se aporten ideas y, por qué no, se creen grupos de programación «internacionales» que puedan dar el fruto deseado en un futuro. Cada miembro de **NET YAROZE** tiene derecho a incluir su página WEB en el servidor de **Sony**, lo que le permitirá mostrar sus creaciones a los demás miembros de la red. Este mes os traemos tres sencillos ejemplos de lo que podréis encontrar en ella. En el número que viene os contaremos más secretos.



LA PAGINA WEB

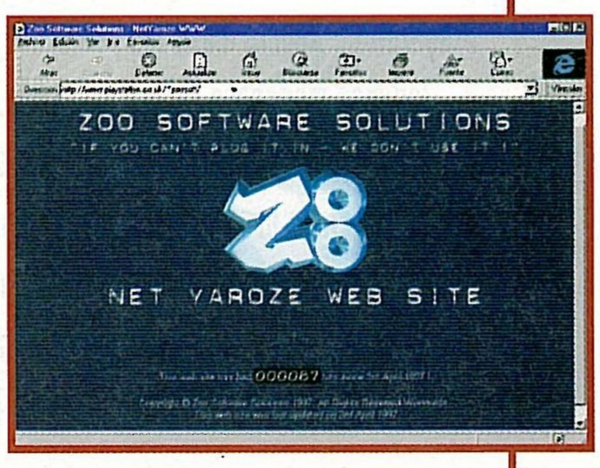
UNA VUELTA POR JAPON

La WEB de **NET YAROZE** en **Japón** lleva funcionando desde Mayo del 96. Podemos acceder a ella a través de un *mirror* existente en la WEB Europea, aunque para sacar algo en claro deberemos echarle un poco de paciencia (por aquello de que todo está en japonés). De cualquier forma, las *demos* japonesas están incluidas en el apartado *demos* de la WEB europea. En breve se pondrá en marcha la página norteamericana, cerrando el triángulo que **Sony** se había propuesto.



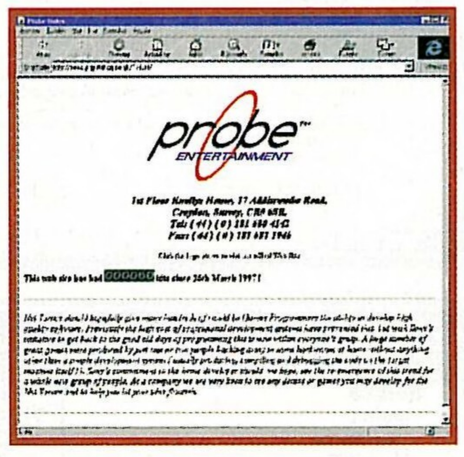
LOS MIEMBROS DE LA WEB

Cada miembro de la WEB tiene derecho a crear su propia página en el servidor de **Sony**. En ella puede mostrar sus creaciones, ideas o proyectos e, incluso, puede fundar su propia «compañía», como ha hecho este miembro de cuya página os mostramos una captura. Si somos realmente buenos, no tardaremos en hacernos muy famosos dentro de la WEB.



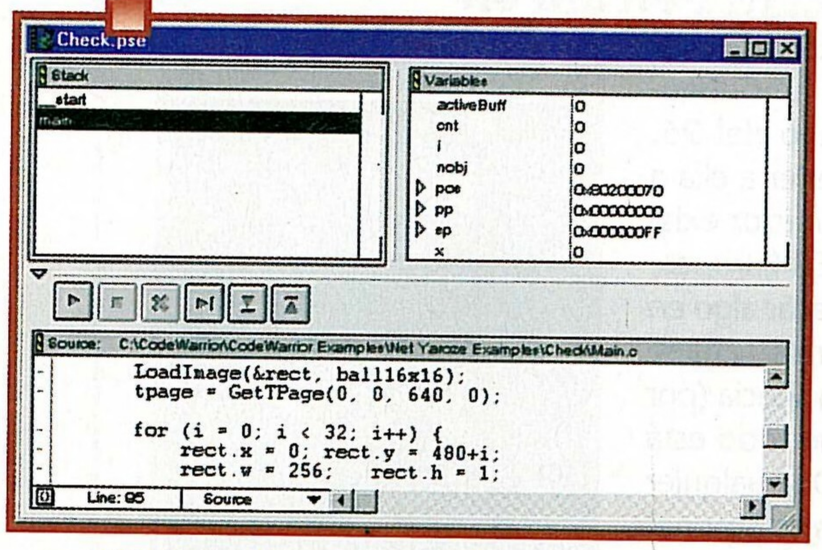
¿BUSCAS TRABAJO?

Si realmente sois buenos en esto de la programación, es posible que vuestro futuro laboral se vea bastante más claro. Esta que tenéis a la derecha es la página que **Probe** mantiene en la WEB de **NET YAROZE**. En ella se pide que los programadores y grafistas les manden sus creaciones, en busca, claro está, de un nuevo genio que engrose sus elitistas filas.

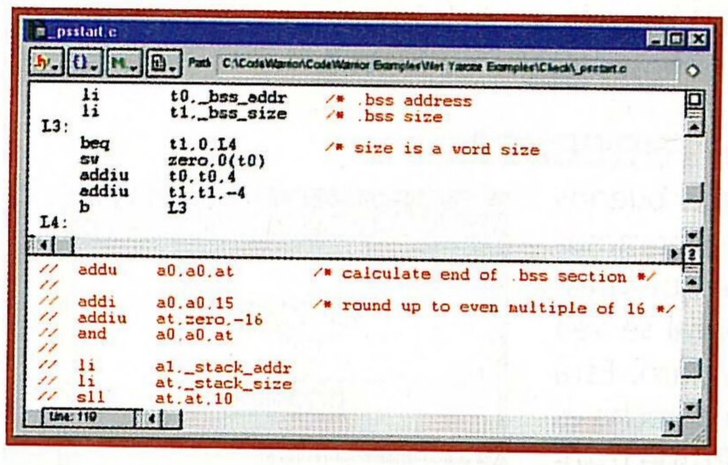


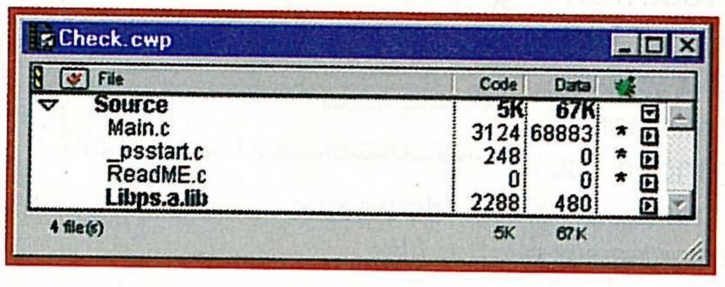
aprende a
PROGRAMAR
con net yaroze

CODE WARRIOR



CODEWARRIOR es un entorno integrado de desarrollo (IDE) para NET YAROZE creado por la compañía Metrowerks. Ha sido diseñado para Windows 95/NT y Mac OS (con Power PC), por lo que los usuarios de este último sistema tendrán también la posibilidad de desarrollar para PlayStation. Aporta todas las posibilidades que incluyen los más modernos IDE del mercado, entre los que se encuentra el ensamblador en línea, el gestor de proyectos o el class browser, elemento imprescindible en la programación en C++. Más información en la dirección http://www.metrowerks.com/products. Y todo ello a un precio bastante más asequible de lo normal.

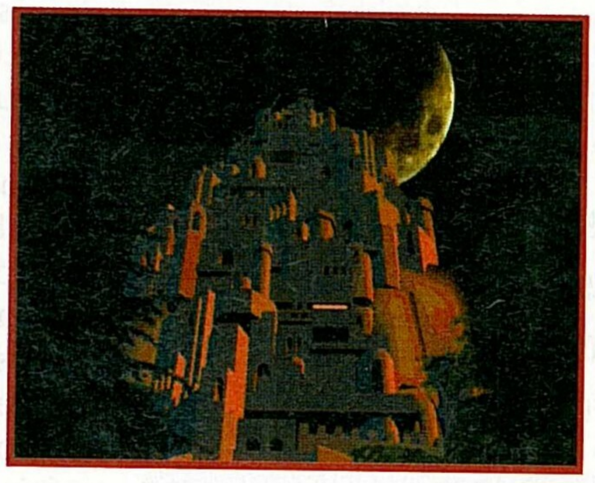




LAS HERR

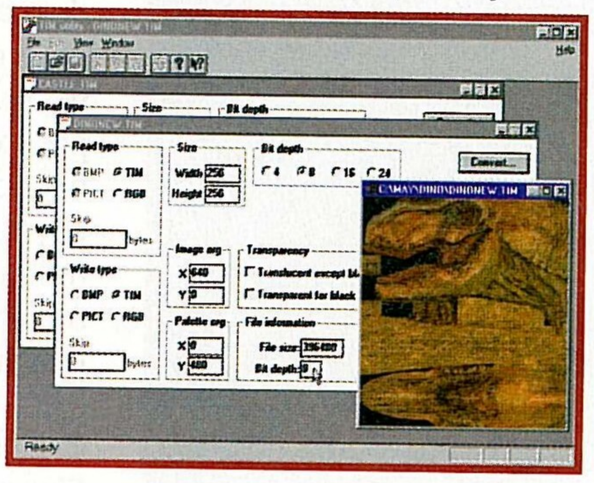
TIMV.BAT

Aplicación que permite visualizar en la televisión, a través de ***PSX***, ficheros TIM, formato gráfico utilizado por las librerías de **NET YAROZE**. Próximamente explicaremos este formato.



TIMUTIL.EXE

Además de realizar otras muchas tareas (que explicaremos más adelante, cuando sea necesario) TIMUTIL.EXE permite la conversión de gráficos entre los formatos TIM, BMP, PICT y RGB.

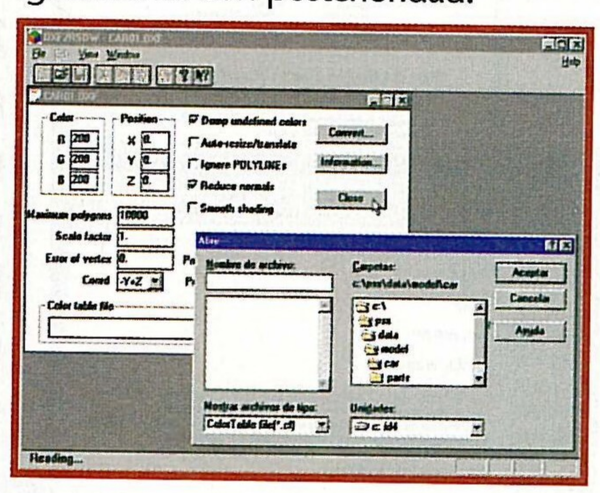


DXF2RSD.EXE

Una vez elaborados los esqueletos de nuestros modelos en 3D, será necesario almacenar éstos con el formato del estándar DXF. Después se debe utilizar este DXF2RSD.EXE, que creará los cuatro ficheros que conforman el formato RSD, utilizado por las librerías 3D de **NET YAROZE**. Cuando entremos a estudiar la programación 3D, se estudiará en profundidad esta aplicación.

DXF2RSDW.EXE

Como se puede imaginar, esta es la versión para ***Windows*** del programa DXF2RSD.EXE que se comentaba en el apartado anterior. Su estudio se verá igualmente con posterioridad.



RSD2DXF.EXE

Crea un fichero DXF (utilizable con la mayor parte de los programas de diseño 3D) a partir de los cuatros ficheros que conforman el estándar RSD. Cuenta con opciones adicionales a través de las cuales podremos recibir información sobre los modelos.

RSDCAT.EXE

Permite la creación de un único fichero RSD a partir de varios ficheros RSD. Su utilización se complementa con el uso de la utilidad RSDFORM.EXE, que se explica en el siguiente apartado. Apenas cuenta con una opción en la línea de comandos, con la que obtendremos información de la conversión.

RSDFORM.EXE

Es, posiblemente, una de las utilidades más complejas de las que se entregan. Permite el movimiento y la transformación de objetos en 3D (en formato RSD). La complejidad de uso radica en la necesidad de indicar las nuevas coordenadas a través de la línea de comandos, sin una referencia visual.

Las herramientas que se entregan con el kit son el punto de partida de nuestra aventura. No os preocupéis si no entendéis del todo lo que se cuenta sobre ellas, en posteriores entregas se explicará con detalle la utilidad de cada una de ellas, según vaya siendo necesaria su comprensión.

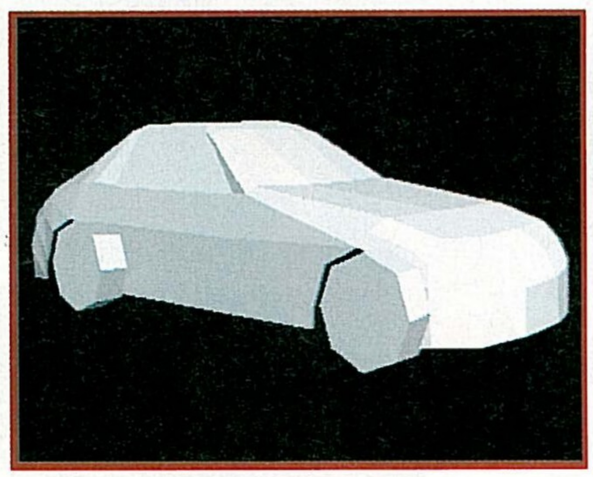
AMIENTAS

RSDLINK.EXE

El formato RSD utiliza cuatro ficheros para almacenar toda la información concerniente a cierto objeto u objetos 3D. Con esta utilidad se unifica en un solo fichero TMD (el formato que finalmente se utilizará) toda la información, con lo que resultará mucho más sencillo su manejo a la hora de crear las aplicaciones. Su utilización es relativamente compleja, por lo que será necesario un estudio a fondo.

RSDV.BAT

Cuando ya se tiene un objeto 3D en formato RSD, se puede utilizar esta utilidad para ***PlayStation*** que permite visualizar en el televisor una *preview* del modelo que se ha creado.



SMF2SEQ.EXE

Entramos de lleno en las utilidades destinadas al manejo del sonido. Esta, en concreto, transforma un fichero estándar MIDI (SMF) en formato SEQ, que es el tipo de fichero utilizado por las liberías de audio de ***PlayStation***.

AIFF2VAG.EXE

Ya que se crean los ficheros de audio con otro tipo de programas, es necesario realizar una conversión entre los formatos existentes. Esta utilidad permite convertir entre los formatos WAV de ***Windows***, PCM, AIFF y SEQ

MKVAB.EXE

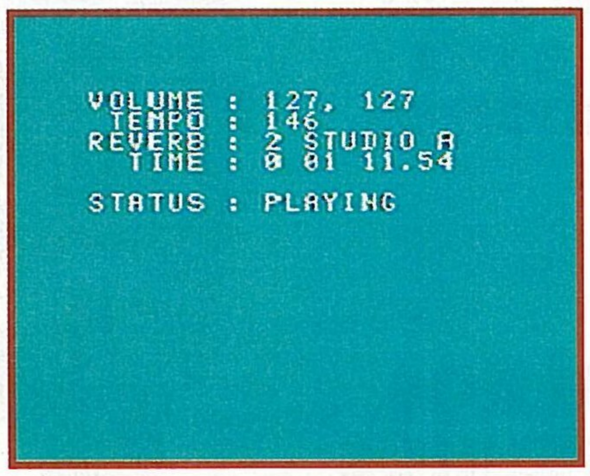
Construye un fichero VAB a partir de las tablas de atributos del formato VAG, que previamente se habrá obtenido con el uso de la utilidad AIFF2VAG.EXE

VABSPLIT.EXE

Crea ficheros separados para las tablas de atributos (VH) y de ondas (VB) a partir del formato VAB. Todo esto será explicado con posterioridad.

SEQPLAY.BAT

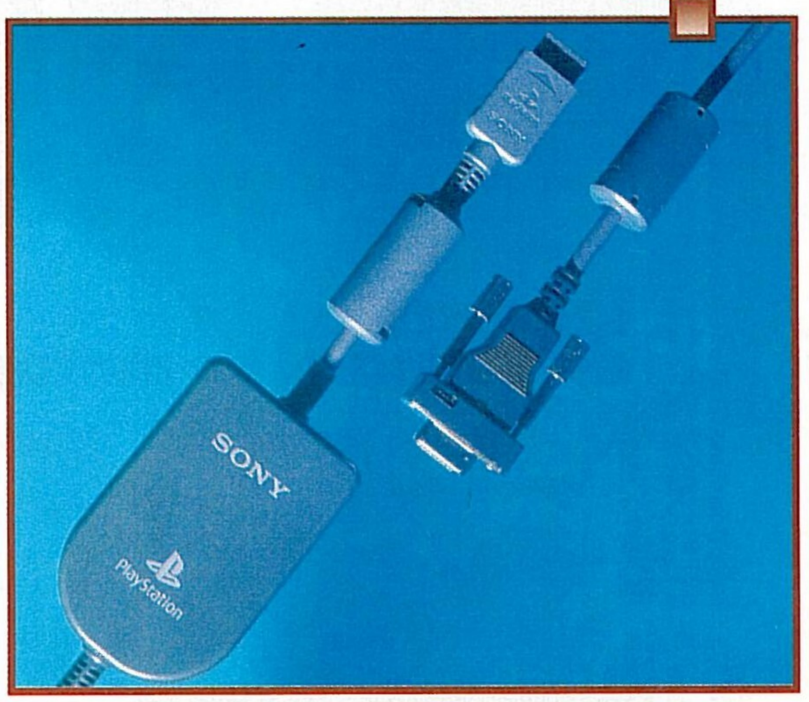
Como su propio nombre indica, permite reproducir un fichero SEQ a través de ***PlayStation***, utilizando las fuentes de sonido incluidas en el disco de arranque de **NET YAROZE**.



VABPLAY.BAT

Al igual que el anterior, aunque en esta ocasión se permite la especificación de las fuentes de sonido que previamente se hayan creado, y no las que por defecto se entregan con el disco de arranque de **NET YAROZE**. Destacar que podremos experimentar con los distintos efectos que el *hardware* de ***PlayStation*** nos permite, lo que ayudará a elegir con mayor exactitud los arreglos que se le introducirán a las músicas. Aunque el sonido no es uno de los puntos más importantes, en un futuro intentaremos tratar este aspecto de la programación.

COMM CABLE



Es sin duda el elemento más importante del kit. Con él se establece la comunicación entre el ordenador y la consola, lo que permite la transferencia y depuración de los programas que creemos. El gestor de comunicaciones para PC se llama SIOCONS y se entrega con el kit. Debido a su vital importancia, hemos decidido dejar para el mes que viene la explicación de esta utilidad.

NUESTRO PRIMER PROGRAMA

ESTE PEQUEÑO PROGRAMA ESCRIBE "HELLO WORLD" EN LA PANTALLA DE NUESTRO PC. AUNQUE ESTO PAREZCA POCO UTIL, EN REALIDAD RESULTA DE MUCHA UTILIDAD, SOBRE TODO CUANDO PROCEDAMOS A DEPURAR NUESTRAS FUTURAS CREACIONES.

```
//*****************************
  * Hellow.c                  *
  * Programa: Hello World!    *
  * Autor: Super Juegos       *
  *****************************//

#include <libps.h>

void main
{
    printf("\nHello World!\n");
}
```

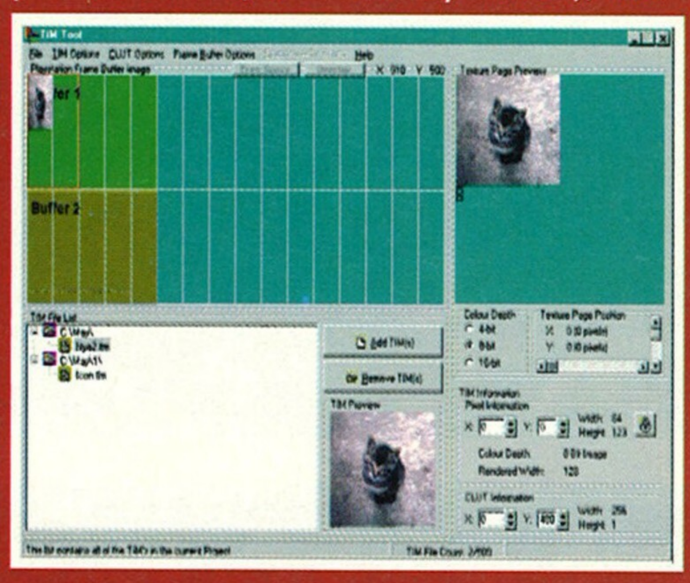
por J. C. MAYERICK

PLAYSTATION

Antes de meternos de lleno en el ámbito de la programación, hemos querido contaros algunas cosas más sobre el Kit de desarrollo, hablando del programa SIOCONS.EXE, una imprescindible utilidad que nos servirá para comunicar **PlayStation** con el **PC**. Comenzamos además el prometido curso de C que el mes pasado avanzábamos, mostrando los distintos sistemas de numeración que serán útiles en la programación. Por último, damos un breve repaso al interior de **PlayStation**, explicando cada uno de los elementos que la componen. Nos gustaría tener más páginas a nuestra disposición, pero no tenemos más remedio que emplazaros al próximo mes para enseñaros algunas cosillas más sobre el intrigante mundo de la programación.

TIMTOOL.EXE

Uno de los defectos del Kit de desarrollo ***Net Yaroze*** radica en lo artesanal del entorno, pero por suerte, **Sony** está creando interesantes aplicaciones que, bajo el entorno ***Windows***, harán que nuestro trabajo sea mucho más sencillo. La estructuración del *frame buffer*, por ejemplo, es una de las tareas más laboriosas y liosas de cuantas se pueden acometer. Para hacer un poco más sencilla esta tarea, **Sony** ha creado esta herramienta, que nos permite organizar toda la información del *frame buffer* de una forma estructurada, desde los *sprites*, hasta las texturas, pasando por los CLUT (*Color Look Up Tables*), etc.



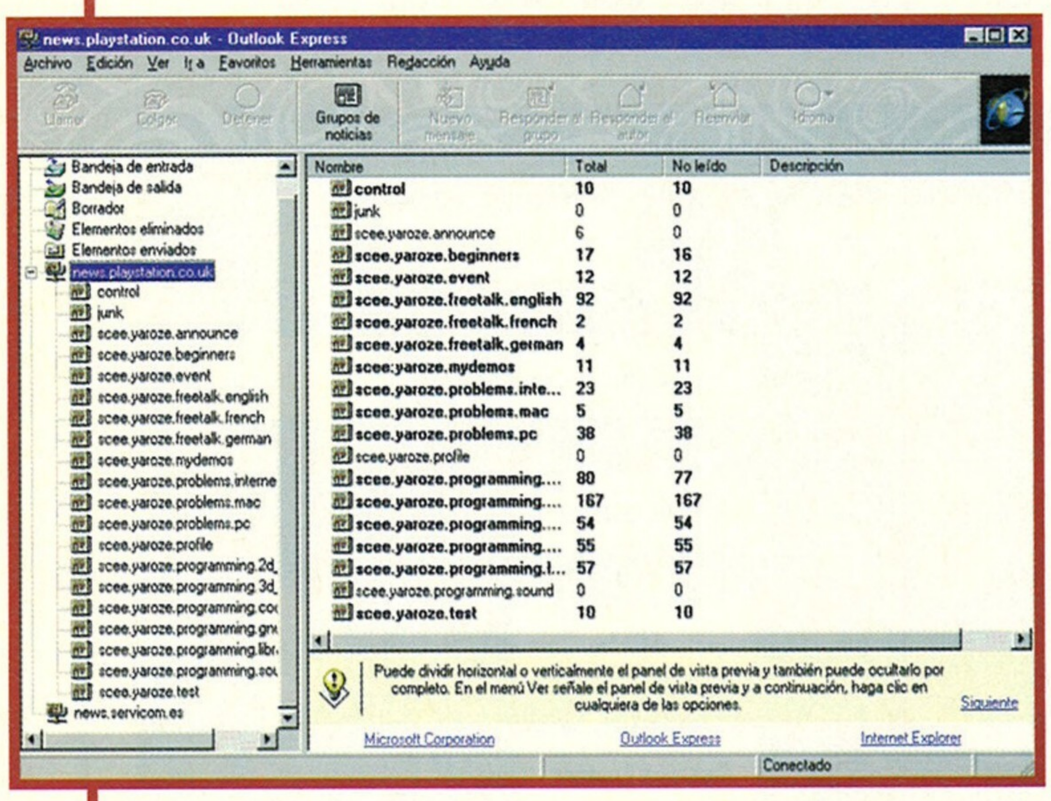
RSDTOOL.EXE

Por suerte, aplicar las texturas a nuestros modelos ya no será tan complicado. Ahora, gracias a esta utilidad diseñada por la gente de **Sony**, sólo tenemos que cargar la textura (en formato TIM) y pinchar en el polígono sobre el que queremos aplicarlo. Así de sencillo. Para obtener una idea más cercana a lo que deseamos, se puede seleccionar una fuente de luz por defecto y un fondo a la pantalla. Si estás interesado en conseguir RSDTOOL.EXE o TIMTOOL.EXE, tan sólo tienes que acercarte a *Links & Utilities* y bajarte la última versión que **Sony** haya programado.

aprende a PROGRAMAR con net yaroze

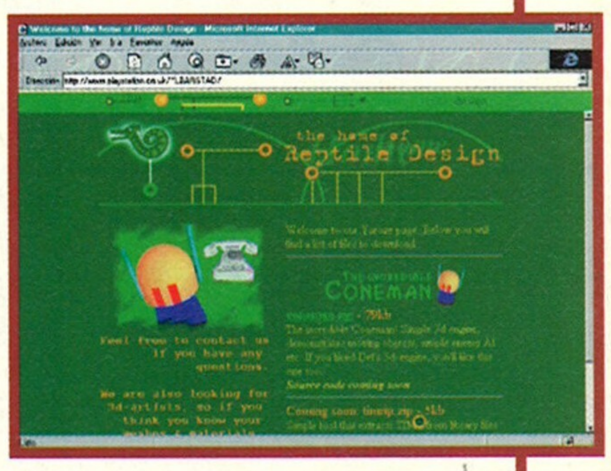
La página WEB está que arde, y el trabajo se nos amontona. Cuando comenzó la aventura con **Net Yaroze** el mes pasado, no se nos pasó por la cabeza que el invento fuese a funcionar así, pero el caso es que nos ha desbordado. Aún así, hemos seleccionado aquello que nos ha parecido más interesante, centrándonos especialmente en los grupos de noticias, una interesante característica de **Internet** que estábamos impacientes por contaros. Desde aquí, además, queremos dar la más sincera bienvenida a **Net Yaroze America**.

LA PAGINA WEB



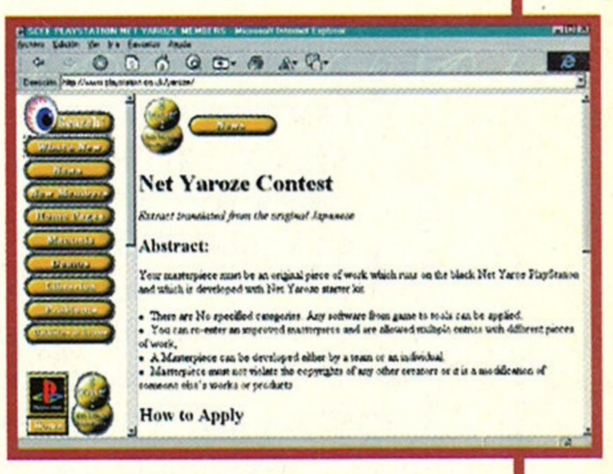
■ LOS 40 PRINCIPALES

La *WEB* de ***Net Yaroze*** está funcionando a la perfección. A día 14 de Mayo de 1997, son ya más de cuarenta las páginas *WEB* de los miembros del *SDK*, lo que augura un excelente futuro al sistema de desarrollo. Entre ellos hay pequeñas joyas como la que os mostramos en la pantalla de la derecha. Su dirección es: http://www.playstation.co.uk/~LBARSTAD



■ CONCURSO NET YAROZE

La revista japonesa de videojuegos DENGEKI-OH ha organizado un concurso mundial de programación de videojuegos. Para participar tan sólo se debe presentar el trabajo (programado con ***Net Yaroze***) antes del 30 de Septiembre de 1997. Los premios son bastante suculentos, y el ganador será elegido por cinco especialistas del sector. Más información en la *WEB* de ***Net Yaroze***, en la sección News.



■ AMERICA YA ESTA ON-LINE

El triángulo ya se ha cerrado. Primero fue **Japón**, después **Europa** y ahora, por fin, **Estados Unidos**. ***Net Yaroze América*** es ya una realidad a la que se puede acceder desde *Utilities & Links*. Pasearse por ella puede ser interminable, ya que desde el 14 de Abril en que se puso en marcha, son ya muchas las decenas de usuarios que han introducido su página personal en la *WEB*. Además, hay que reconocerlo, es una página muy, muy completa.

■ LOS GRUPOS DE NOTICIAS

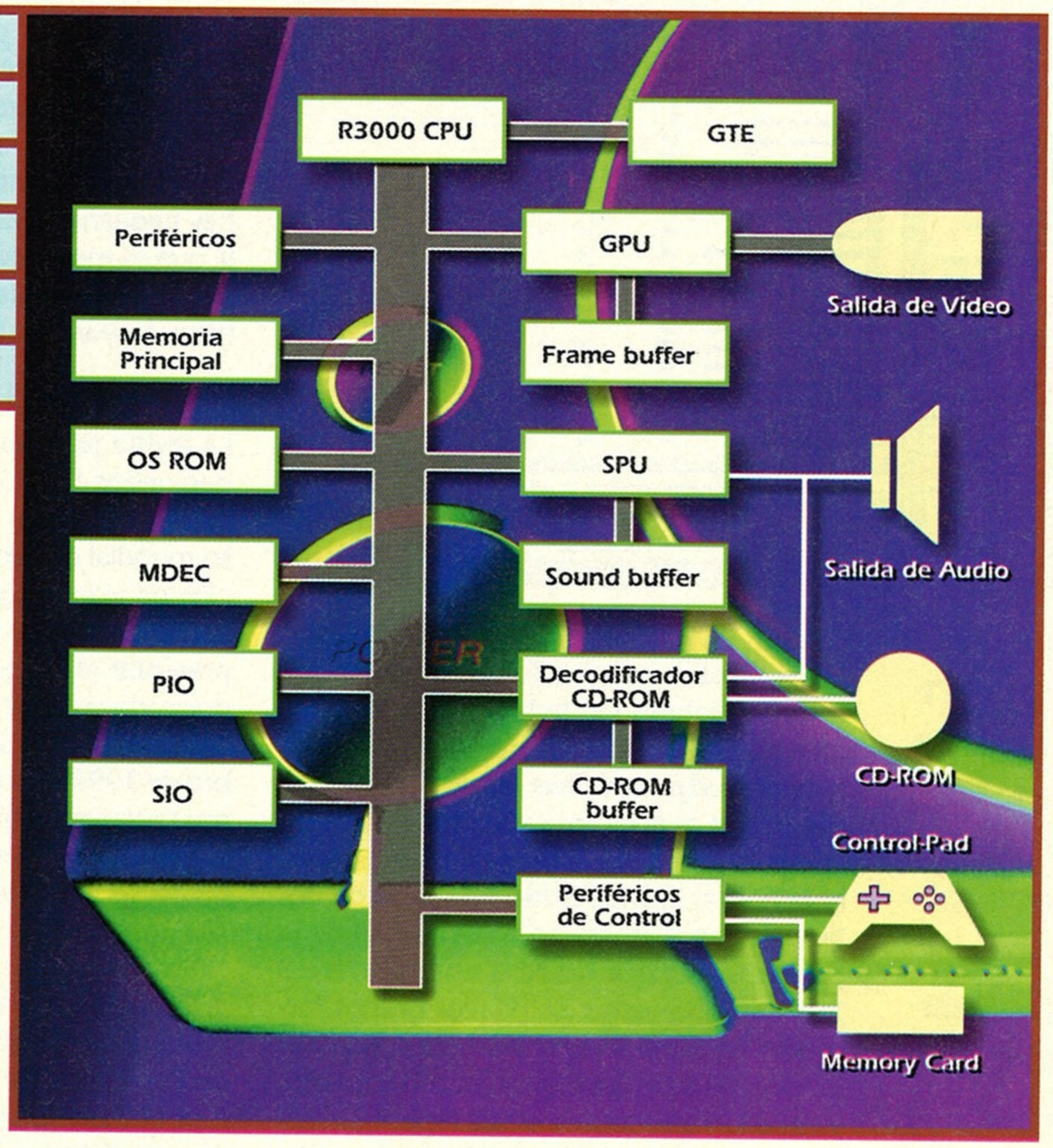
Si eres miembro de **Net Yaroze** y todavía no has hecho uso de los grupos de noticias, es el momento de hacerlo. En ellos encontrarás toda la ayuda posible.

Los que estén más familiarizados con la red **Internet** sabrán ya de la utilidad de estos grupos. En ellos son los propios usuarios los que se ayudan a resolver las distintas dudas, con lo que se acrecenta aún más ese aire de compañerismo que la gente de **Sony** quiere imponer en la *Web*. Existe un grupo de noticias (**news.playstation.co.uk**) oficial de la compañía, aunque desde el lanzamiento del *SDK* han sido varios los nuevos grupos formados en torno a la programación con ***Net Yaroze***. Su funcionamiento es muy sencillo, ya que tan sólo tenemos que especificar como nuevo servidor de noticias la dirección arriba indicada. Una vez hecho esto se puede optar por una de estas tres posibilidades: una, investigar cada uno de los mensajes disponibles en el servidor en busca de un tema que nos interese; dos, enviar alguna duda con la esperanza de que alguien nos responda en los siguientes días; y tres, ayudar a otro usuario resolviendo la dificultad o duda que haya planteado. Como se puede ver, los grupos de noticias serán la mejor ayuda que se pueda disponer.

aprende a
PROGRAMAR
con net yaroze

EL INTERIOR DE LA PLAYSTATION

El interior de PlayStation encierra gratas sorpresas para los futuros programadores. Todo está pensado para ahorrar trabajo al diseñador. Como se verá en futuras entregas, la funcionalidad de algunos de los componentes de la consola pueden llegar sobrepasar de lleno lo que de ellos podíamos esperar. Todo, en fin, parece más sencillo.



Este esquema muestra, de una forma bastante sencilla, la organización interna de una ***PlayStation***. En lo alto del gráfico se encuentra la CPU, un procesador RISC (*Reduced Instruction Set Code*) de 32 *bits* a 33Mhz basado en el R3000. Este procesador cuenta con 32 registros y una I Caché de 4 Kbytes integrada en el mismo chip. A su lado se encuentra el GTE (*Geometry Transfer Engine*) que se encarga de realizar las operaciones aritméticas con matrices y vectores, y cuyos resultados pueden ser utilizados posteriormente, y sin ninguna modificación, por la GPU. Esta última (*Graphics Processing Unit*) realiza el trabajo fuerte en cuanto a gráficos. Esta unidad se encarga del trabajo concerniente a *sprites*, polígonos, *scroll*, fondos, etc. Para ello necesita de la inestimable ayuda del *frame buffer*, una memoria de vídeo de 1 MByte que sirve para almacenar toda la información referente a gráficos, incluido, por ejemplo, las CLUT (*Color Look Up Table*), paletas de color que posteriormente serán utilizadas por *sprites*, polígonos, etc. La GPU junto con el *frame buffer* será, a partir del mes que viene, el centro de atención de esta sección.

PlayStation cuenta con 2 MBytes de memoria principal, así como con 512K de Sistema Operativo en ROM. La SPU (*Sound Processing Unit*) cuenta con un *sound buffer* de 512 Kbytes y 24 voces, y está intimamente ligado al decodificador de CD-ROM, encargado de leer la información desde el CD. El PIO y el SIO son los puertos de expansión paralelo y serie respectivamente, mientras que el MDEC es un dispositivo especial destinado a la compresión/descompresión a gran velocidad de JPEG o MPEG, aunque esta característica no se incluye inicialmente en el Kit de desarrollo ***Net Yaroze*** (pero sí en el profesional). Por último, y aunque no aparezca en el gráfico, está el DMA, que es el encargado de transferir los datos entre la memoría principal y los distintos dispositivos (*frame buffer*, *sound buffer*, etc.) El mes que viene entraremos de lleno en el intrincado mundo de la generación de gráficos.

La programación en C es la base de la programación con el Kit de desarrollo de **Sony**. Desde este mes os brindamos la posibilidad de aprender este lenguaje de programación, uno de los más utilizados en todo el mundo, de una forma sencilla y clara. Aquí se mostrará la teoría, pero la práctica llegará al tiempo que profundicemos en el funcionamiento del Kit.

C: SISTEMAS DE NUMERACIÓN

El sistema numérico utilizado actualmente es la numeración decimal o en base 10. Existen tres tipos de numeración más que nos serán de gran utilidad en el estudio del C, como son el binario, el octal y el hexadecimal. El primero, en base 2, se representa con los digitos 0 y 1, mientras que en el octal se utilizarán los dígitos del 0 al 7, ambos inclusive. El más importante de todos, el hexadecimal, necesita de 16 dígitos para su representación. Por un lado se utilizan, al igual que en la numeración decimal, los dígitos del 0 al 9, a los que se debe añadir las letras A a la F, completando así los 16 dígitos requeridos. Dichos carácteres representan los valores comprendidos entre 10 y 15. Así, la A equivale a 10 (en sistema decimal), la B a 11, y así hasta la F, que equivale a 15. Aunque es conveniente dominar el sistema hexadecimal, es posible representar en base 10 cualquier tipo de número, sea cual sea su base. Esta es la fórmula que se debe utilizar para pasar cualquier tipo de número a su equivalente en base 10 o sistema decimal:

- $(n0)b^0+(n1)b^1+...+(nx)b^x$

La b es la base de numeración y nx el dígito. Así, el número 10110b, en base binaría, equivaldría en decimal a:

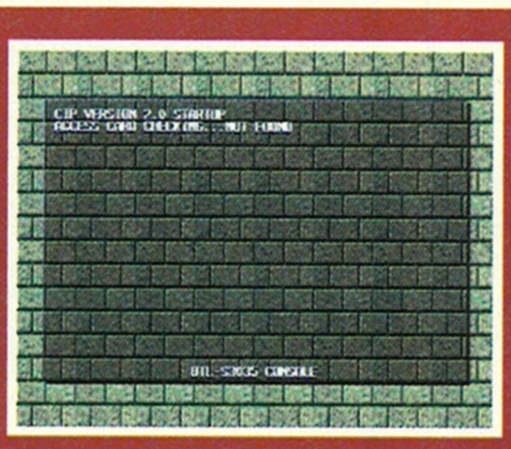
- $0*2^0+1*2^1+1*2^2+0*2^3+1*2^4$, lo que equivale a 0+2+4+0+16 = 22 decimal.

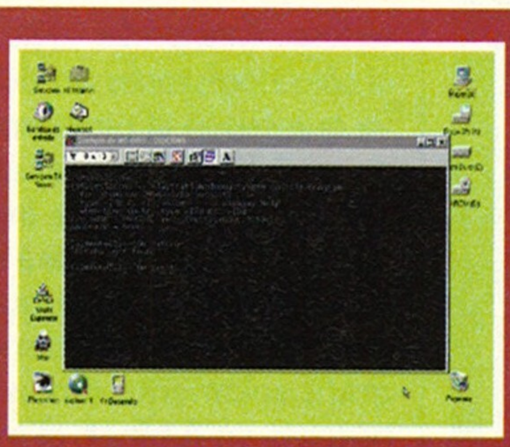
Ahora, el número 2AF1h hexadecimal se representaría en base 10 como:

- $1*16^0+15*16^1+10*16^2+2*16^3$, que simplificando queda de esta forma: 1+240+2560+8192 = 10993 decimal.

La operación inversa es quizás más fácil. Primero se divide por la base a la que se quiere convertir; después se hace lo mismo con el cociente, y así hasta que el dividendo sea inferior al divisor. Tomemos el numero 10231d que queremos pasar a hexadecimal. Hacemos: 10231/16 que queda como 639 de cociente y 7 de resto. Ahora, 639/16 nos da 39 como cociente y 15 de resto. Por último, 39/16 nos deja 2 como cociente y 7 de resto. Ahora sólo nos queda colocar sucesivamente el último cociente y el resto de cada operación, quedando 27F7h. Muy fácil.

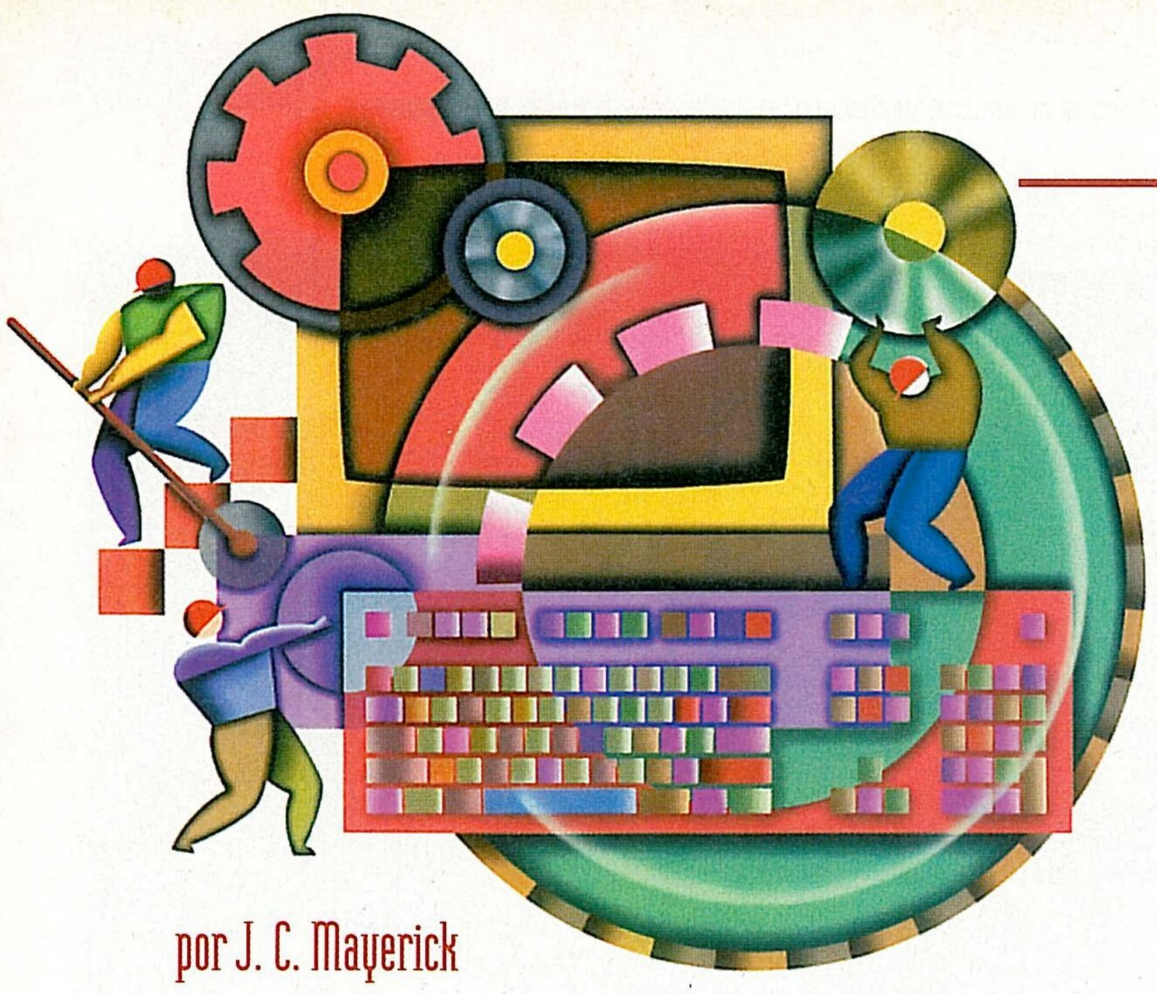
Como habréis notado, detrás de cada número se coloca una letra que identifica la base en que éste está representado: b para binario, h para hexadecimal, q para octal y d para decimal. Cuando no se incluye el especificador, se supone en base 10. El més que viene os mostraremos algo más sobre numeración y os enseñaremos los tipos de datos.





SIOCONS.EXE

En la comunicación entre el **PC** y la **Net Yaroze** entran en juego tres elementos. Por un lado está el cable serie, por otro el sistema operativo de la consola, que se entrega en CD, y por último el programa SIOCONS.EXE, una aplicación que permite la transferencia de datos entre ordenador y consola. Su funcionamiento, que viene extensamente explicado en el capítulo 17 de la guía del usuario, no es excesivamente complicado, y en apenas unos minutos seremos capaces de transferir las demos que se encuentran en la página WEB de **Sony** sin demasiados problemas. De cualquier forma, SIOCONS.EXE será especialmente útil en la depuración de programas, ya que desde el se puede explorar la memoria de la consola, los registros del procesador e, incluso, se puede desensamblar parte del código para una mejor comprensión de lo que ocurre durante la ejecución. Dominar esta aplicación se nos antoja imprescindible si se desea progresar lo suficiente.



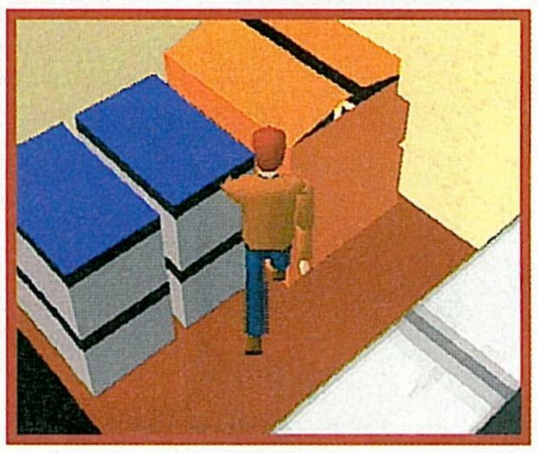
por J. C. Mayerick

No sabemos si será la llegada del verano o qué, pero la cosa comienza a ponerse caliente. Este mes tenemos de todo, desde un curioso juego en 3D llamado YAKATA, hasta un útil *add-on* para PHOTOSHOP. Daremos, además, un pequeño tirón de orejas a la gente de Sony y continuaremos con nuevas entregas de los cursos de programación.

YAKATA:
APRENDIZ DE RESIDENT EVIL

ES LO MEJORCITO QUE HEMOS ENCONTRADO EN LA WEB, Y ADEMAS RESULTA INTERESANTE DE JUGAR.

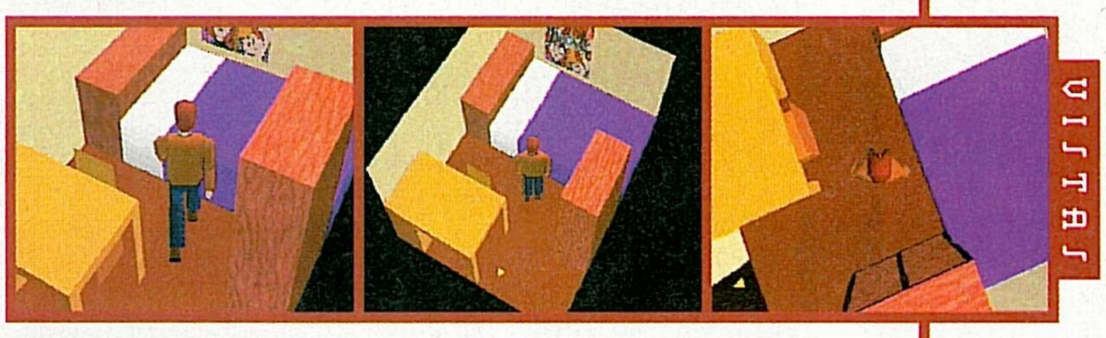
Alucinados nos quedamos en la redacción al observar las buenas maneras de esta *demo* que encontramos en la WEB japonesa. No llega a la calidad de RESIDENT EVIL, ni siquiera a la de ALONE IN THE DARK, pero es un perfecto ejemplo de lo que se puede conseguir con un poco de destreza y, cómo no, el



kit ***Net Yaroze***. Trás una pequeña *intro*, nos adentraremos en una tétrica mansión de la que, suponemos, deberemos escapar. Y decimos «suponemos» porque, a falta de que **R. Dreamer** mejoré su japonés, nos encontramos con el grave inconveniente de no saber que hacer, salvo lo elemental. De cualquier forma, podemos confirmar

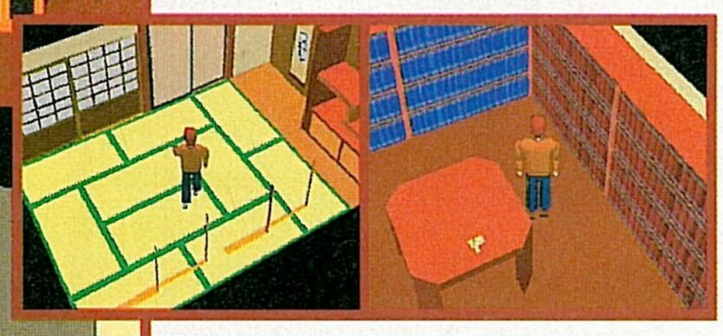
INVENTARIO

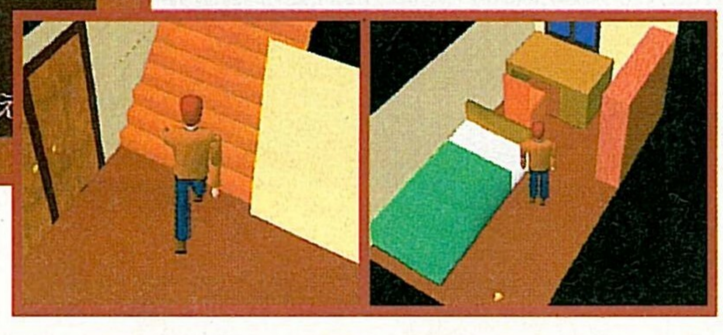
ADEMAS DE CONTAR CON UNOS BUENOS GRAFICOS, PODEMOS DAR FE DE QUE NO SE RALENTIZA.



VISTAS

OBJETOS





que cuenta con un extenso mapeado, dieciséis vistas diferentes y un inventario preparado para albergar decenas de objetos. Fácil, desde luego, no parece. Si alguien progresa más allá de la chimenea que nos lo comunique, quién sabe si algún día deberemos publicar la guía de este curioso juego. Por desgracia, el código fuente del juego no está incluido, por lo que deberemos conformarnos con jugar.

aprende a PROGRAMAR con net yaroze

Moradas las hemos pasado este mes para confeccionar LA PAGINA WEB con el escaso material existente. Aún así, hemos recogido lo más interesante que hemos encontrado en la World Wide Web y lo hemos plasmado en el papel como mejor lo sabemos hacer: con una imágen y mil palabras (para contentar a todos). Sólo nos queda esperar que el mes que viene las pasemos de nuevo moradas, pero por lo complicado de seleccionar lo más interesante de entre toneladas de material. Aprovechad las vacaciones para programar algo...

LA PAGINA WEB

■ PEQUEÑO TIRÓN DE OREJAS

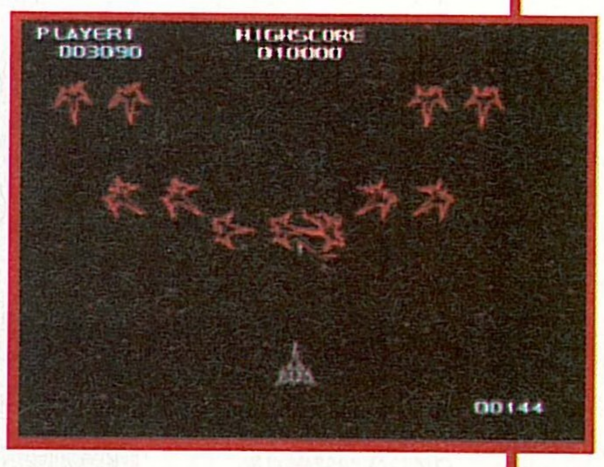
...para **SCEE**. La página *web* de **Net Yaroze Europa** se ha sumido en un incómodo letargo del que no parece despertar. A 12 de Junio de 1997, la última actualización correspondía al apartado *What's New* y databa del día 21 de Mayo de 1997. A este paso, lo único interesante será visitar las páginas de los miembros de la *web*. Esperemos que no se vuelva a repetir...

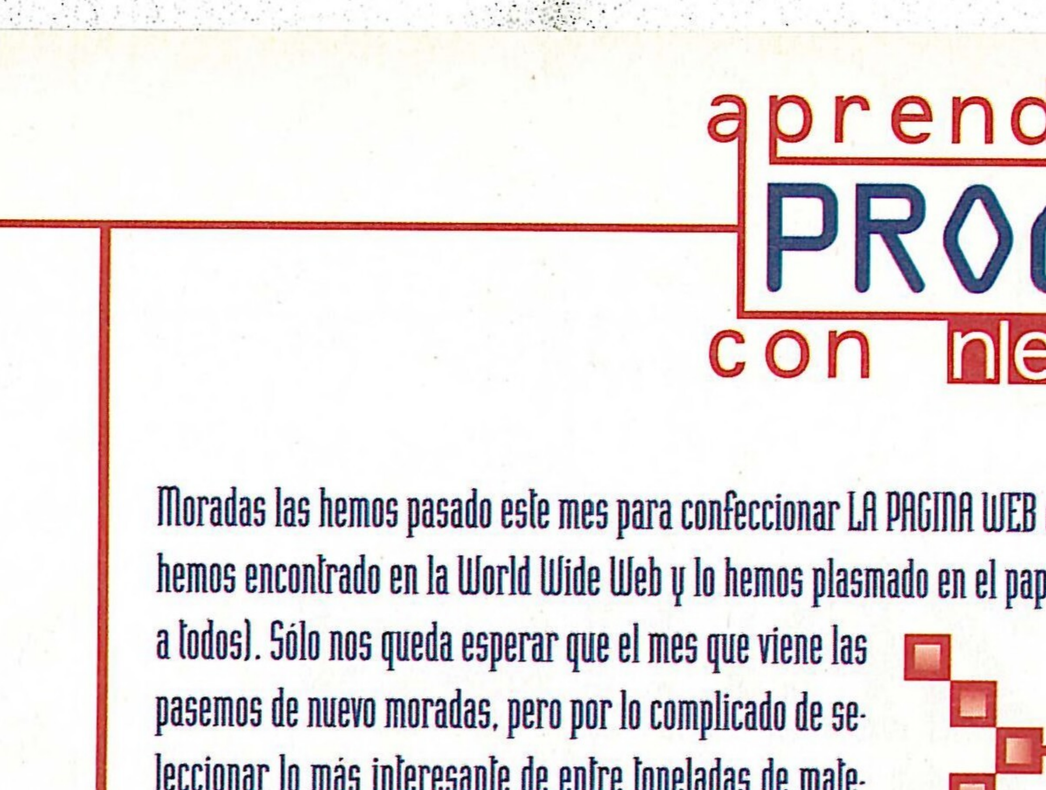
■ EL MEJOR REMEDIO...

A falta del movimiento esperado en la página *web* europea, lo mejor es dejarse llevar por las tentadoras ofertas de las *webs* americana y japonesa. La primera cuenta con decenas de *demos* y, además, con un *interface* ciertamente ameno, mientras que la segunda oferta las *demos* de mayor calidad de las tres. Lástima que sea prácticamente imposible entender nada de lo que éstos nos pretenden contar.

■ FLIPADA VECTORIAL

BILLS 110 es un interesante *shoot'em-up* (que podréis conseguir en **Net Yaroze América**) con gráficos vectoriales. Para crearlo, su programador ha utilizado la posibilidad de crear líneas con degradados que el *hardware* de la ***PlayStation*** proporciona, consiguiendo unos resultados ciertamente curiosos. Cuenta con cinco niveles que podremos configurar a nuestro antojo modificando el código fuente que su creador ha incluido en el fichero ZIP.





Relizar la conversión desde cualquier formato gráfico al utilizado por el kit Net Yaroze ya no será ningún problema gracias a este add-on para PHOTOSHOP.

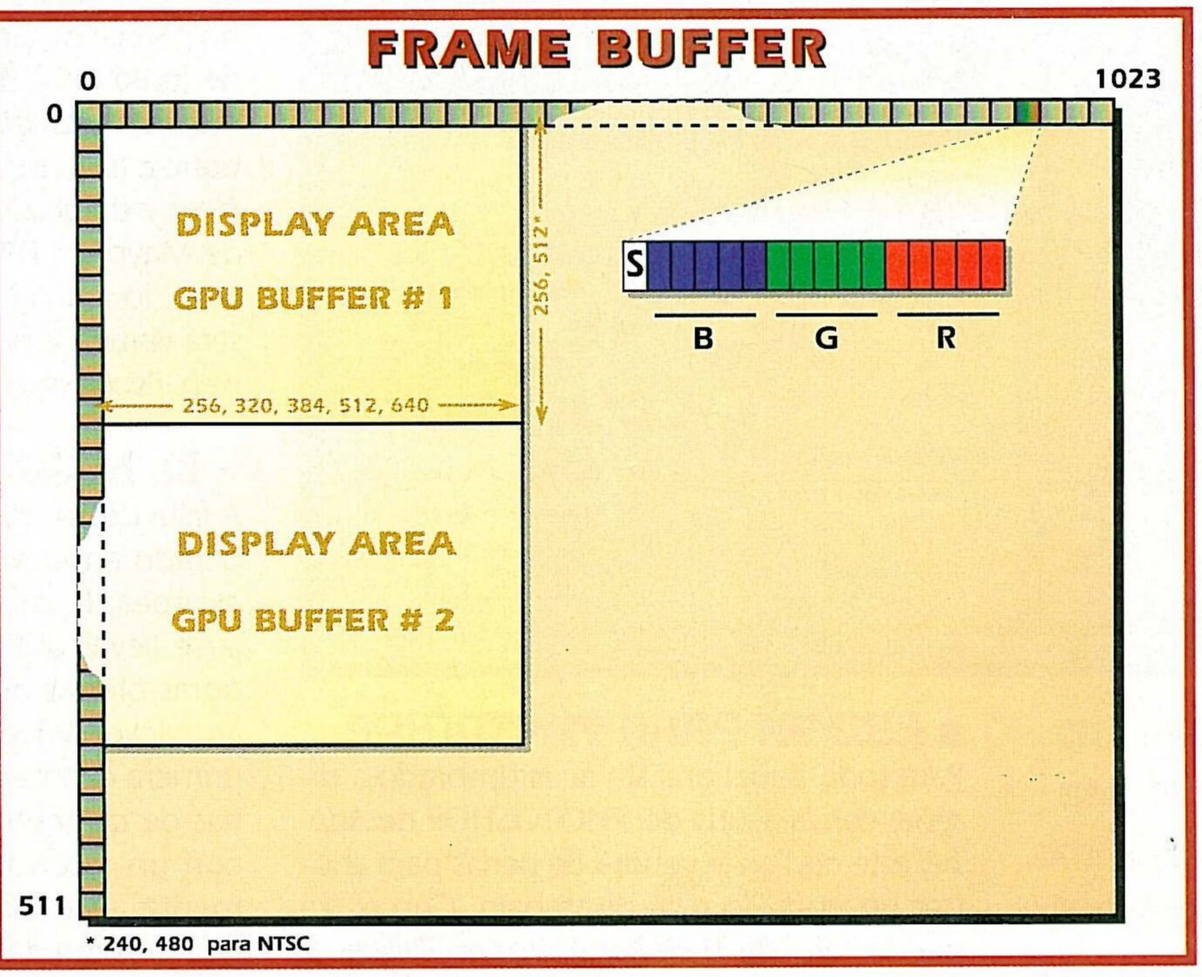
■ ADD-ON PARA PHOTOSHOP

Para todo aquel grafista acostumbrado a diseñar con la ayuda del PHOTOSHOP de ***Adobe***, este *add-on* le vendrá de perlas para ahorrar un poquillo más de trabajo. Con él (se puede bajar de *Third Party Links* en *Utilities & Links*) no será necesario realizar el incómodo cambio de formato gráfico a través de TIMUTIL, la utilidad suministrada por **Sony**. De esta forma, al grabar una imágen desde PHOTOSHOP en formato TIM, aparecerá un cuadro de diálogo sobre el que se podrán definir los distintos parámetros que este formato gráfico requiere. Por un lado, se debe especificar el tamaño del CLUT (*Color Look Up Table*) entre 4 y 8 *bits*, o bien la utilización de *pixels* de 16 o 24 *bits*. Por otro, la posición y el tamaño en la *VRAM*, tanto de las tablas de color (CLUT) como de la propia imágen. Por último, será necesario especificar el redondeo de los *bits* bajos de cada *pixel*, así como la interpretación que se realizará del negro, bien traslúcido, bien trasparente. Sin duda una gran noticia de la que, a pesar de llevar bastante tiempo en la *web*, no habíamos podido informar hasta ahora.

aprende a
PROGRAMAR
con nelt yaroze

ORGANIZACION DE LA VRAM

Es posible que trás leer el apartado que hoy le dedicamos al FRAME BUFFER muchos encuentren poco útil su explicación. No se darán muchos detalles técnicos, ni siquiera una breve explicación que tenga que ver con la programación. Si hablamos del FRAME BUFFER es porque su conocimiento resultará imprescindible en un futuro.



Programar para ***PlayStation*** puede suponer un pequeño trauma para aquellos programadores que hayan experimentado con otro tipo de *hardware* diferente. Se ha pasado de los omnipresentes bloques de 8x8 *pixels* que se venían utilizando desde las primeras ***coin-op*** de finales de los 70 (y que aún ***Super Nintendo*** y ***Mega Drive*** utilizaban) a un espacio de memoria de 1 *Mbyte* sobre y desde el que se puede escribir y leer a nuestro antojo, aunque eso sí, siempre a través del DMA. De cualquier forma, y si estamos muy familiarizados con el modo de programación por bloques tampoco hay que preocuparse, ya que como se verá en próximos números existe la posibilidad de utilizar dicho sistema.

El FRAME BUFFER (o la VRAM, como se prefiera) cuenta con una tamaño original de 1024 x 512 *pixels*, lo que supone 512 *Kbytes*. Teniendo en cuenta que el modo de representación por defecto es el de color directo de 16 *bits* (2 *bytes*) por *pixel*, se consigue la cifra de 1 *Mbyte*, tamaño total de la VRAM.

Se pueden seleccionar cuatro tipos distintos de representación: 4, 8, 16 o 24 *bits* por *pixel*, consiguiendo 16, 256, 65.536 y 16.777.216 colores respectivamente. En los dos primeros modos se requiere además la utilización de CLUTs (*Color Look Up Tables*), que se almacenan también en la VRAM y que contienen los colores que van a ser utilizados. Puede haber multiples CLUT. Los dos últimos modos no requieren paletas de color ya que, como se muestra en el gráfico, se almacena directamente en la VRAM en formato RGB. El *bit* S sirve para indicar que ese *pixel* será transparente.

Las zonas marcadas como GPU BUFFER #1 y GPU BUFFER #2, muestran las porciones de la VRAM que serán presentadas en pantalla. La existencia de dos *buffers* permite la utilización de la técnica *Double Buffering*, por la cual se permite presentar en pantalla la información de uno de los *buffers* mientras el otro se está generando. Esto evita perder tiempo mientras se espera a que se produzca un nuevo retrazado vertical, periodo durante el cual no se puede presentar su contenido.

Este mes comenzamos con la explicación de los distintos tipos de datos que se pueden utilizar en C. Al ser uno de los apartados más extensos de la programación en este lenguaje, más que nada por la complejidad de uso de algunos de sus componentes (referencias y punteros principalmente), se ha optado por dividir su definición en varias partes.

C: TIPOS DE DATOS

Los tipos de datos es, con diferencia, uno de los aspectos más importantes de la programación en C. Conocer las diferencias existentes entre cada uno de ellos y escoger el tipo correcto en cada caso nos ayudará a generar un código mucho más claro y eficaz. Existen gran variedad de tipos, que serán agrupados según sus características y funcionalidad. Pueden ser constantes y variables, aunque por su importancia nos centraremos en las del segundo grupo. Dentro del grupo de las variables existen tres divisiones claramente diferenciadas: estructuras, variables simples y referencias, apartado este último especialmente importante y complicado que se estudiará próximamente. Lo que realmente nos interesa, por ahora, es comprender el funcionamiento de las variables simples, cuya definición y forma se puede apreciar en el cuadro de la derecha.

```
void main()
{   char letra = "L";
    int entero = 125;
    float real = 12.543;
    printf("letra = %c\n
            entero = %d\n
            real = %f\n",
        letra, entero, real);
}
```

TIPOS DE VARIABLES SIMPLES

VARIABLES ENTERAS

TIPO	TAMAÑO	NOMBRE	RANGO DE VALORES
char	1 byte	carácter	-128 a 127
unsigned char	1 byte	carácter sin signo	0 a 255
int	–	entero	–
unsigned int	–	entero sin signo	–
short	2 bytes	entero corto	-32.768 a 32.767
unsigned short	2 bytes	entero corto sin signo	0 a 65.535
long	4 bytes	entero largo	-2.147.483.648 a 2.147.483.647
unsigned long	4 bytes	entero largo sin signo	0 a 4.294.967.295
enum	2 bytes	entero corto	-32.768 a 32.767

VARIABLES DE COMA FLOTANTE (NUMEROS REALES)

TIPO	TAMAÑO	NOMBRE	RANGO DE VALORES
float	4 bytes	simple precisión	3.4E ± 38
double	8 bytes	doble precisión	1.7E ± 308
long double	10 bytes	largo doble precisión	1.2E ± 4.932

El primer punto a tener en cuenta es la declaración. Antes de utilizar una variable es necesario declararla, pero siempre precediendo a la definición de las funciones. Dependiendo del lugar en que sean declaradas variará también el ámbito de la variable (la parte del código en que podrá ser utilizada). Por poner un ejemplo, si una variable se declara fuera de cualquier función, ésta podrá ser utilizada desde cualquier punto del programa, mientras que si se define dentro de una función (que no sea *main*), su funcionamiento se verá reducido al interior de ésta (siempre y cuando no se especifique lo contrario).

Una variable se define como sigue:

```
char nombre [= "Yaroze"];
```

Primero se especifica el tipo de dato, después el nombre y por último el valor o el contenido con que se va a inicializar dicha variable (los corchetes significan que esa parte es prescindible). Si no se inicializa la variable, entonces su contenido sería indefinido.

La elección de la variable a utilizar varía del tipo de dato que deseemos almacenar. Si es un nombre, utilizaremos una variable de tipo *char*; si es un número con muchos decimales, utilizaremos *float*, *double* o *long double*; mientras que si el dato va a ser un número entero, lo mejor será elegir cualquiera de las variables enteras que se ponen a nuestra disposición. Sobre la tabla de variables se muestra un sencillo ejemplo cuyo ejecutable se puede crear con el compilador en línea GCC.EXE.

Una vez ejecutado con la ayuda del programa SIOCONS. EXE, en la pantalla del ordenador aparecerá lo siguiente:

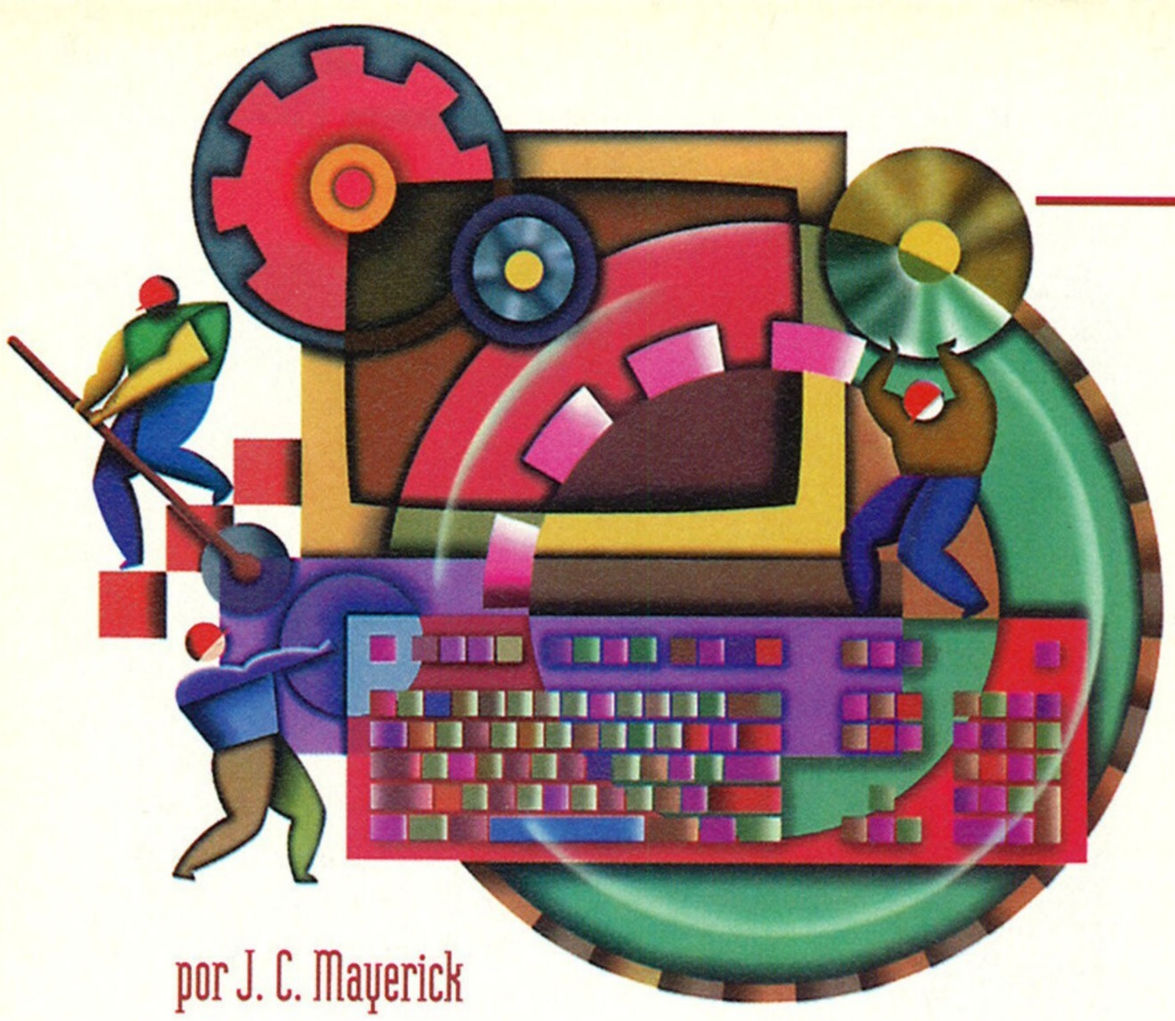
```
letra = L
entero = 125
real = 12.543
```

Es imprescindible incluir el fichero de cabecera libps.h para poder compilarlo correctamente.

En dicho ejemplo se definen tres variables de diferente tipo. Una variable de carácter, un entero y un número real. Con *printf* se muestra el pantalla el contenido de cada una de esas variables. %c, %d y %f indica que en su lugar se impriman los valores de la variables que siguen a las comillas: letra, entero y real. \n inserta los códigos de nueva línea y retorno de carro allí donde aparece.

Este es sólo el aperitivo, el mes que viene continuaremos con la explicación de los tipos de datos, así como su manejo y aplicación.

PROGRAMA HELLO WORLD



por J. C. Mayerick

Pasemos a la práctica analizando el primer programa que todo usuario debe crear cuando se introduce en un nuevo lenguaje de programación. Ese pedazo de listado que aparece a la derecha sólo escribe «Hello World!» en la pantalla. Entre éste y el próximo número conoceréis su funcionamiento.

Aunque en un principio no entendáis absolutamente nada, no os preocupéis. La única finalidad que buscamos hoy es que comprendáis la forma de trabajar a la hora de programar para ***PlayStation.***

Las dos primeras líneas «#include...» indican que se cargen ficheros de cabecera con las librerías de ***PlayStation*** y las rutinas de control del *pad*. En un fichero de cabecera se suelen definir funciones (pequeños programas) que después serán utilizados en el programa principal. Las directivas «#define...» indican que, allá donde aparezca el nombre que le sigue, lo sustituya por el valor (o cadena) que sigue a continuación. Acto seguido se definen las tablas de ordenación. Estás tablas almacenan el orden en que los distintos elementos deben ser dibujados en pantalla, para que unos *sprites* se superpongan a otros, etc. Llegamos al módulo *main*, la función principal que todo programa debe tener y punto desde el cual comienza la ejecución. En él se definen dos variables (curso de C, Nº63) y el modo de pantalla, NTSC o PAL. Se inicializa la pantalla (*ResetGraph*(0)) y se define la resolución del *frame buffer* (512x240). La siguiente línea indica que el primer *buffer* de pantalla va desde las cordenadas XY 0-0 a 255-255. 0-256 se corresponde con el primer *pixel* del *buffer* número dos. Tener dos *buffer* permite trabajar sobre uno mientras el otro está siendo presentado en pantalla. Por último, se carga la fuente que se va a utilizar y se inicializa el *control pad* con *PadInit*(). El resto del listado, lo verdaderamente interesante, será explicado en el próximo número, cuando haya un poquito más de espacio.

```
/***************
 *
 *    Tuto1.c
 *
 ***************/

#include <libps.h>
#include "pad.h"

#define PACKETMAX       (10000)
#define PACKETMAX2      (PACKETMAX*24)
#define OT_LENGTH       (14)

GsOT        WorldOrderingTable[2];
GsOT_TAG    zSortTable[2][1<<OT_LENGTH];

PACKET      GpuOutputPacket[2][PACKETMAX2];

int main (void)
{
        u_long PadStatus = 0;
        int outputBufferIndex;
        SetVideoMode( MODE_NTSC );
//      SetVideoMode( MODE_PAL );

        ResetGraph(0);
        GsInitGraph(512 ,240,
GsOFSGPU|GsNONINTER, 1, 0);
        GsDefDispBuff(0, 0, 0, 256);

        FntLoad(960, 256);
        FntOpen(32, 32, 256, 200, 0,
512);

        PadInit();

        WorldOrderingTable[0].length =
OT_LENGTH;
        WorldOrderingTable[1].length =
OT_LENGTH;
        WorldOrderingTable[0].org =
zSortTable[0];
        WorldOrderingTable[1].org =
zSortTable[1];

        for (;;)
        {
        PadStatus = PadRead();
        if (PadStatus & PADselect)
        break;
        outputBufferIndex = GsGetActive-
Buff();
        GsSetWorkBase(
(PACKET*)GpuOutputPacket[outputBufferIn-
dex]);
        GsClearOt(0, 0, &WorldOrderingTa-
ble[outputBufferIndex]);
        DrawSync(0);
        VSync(0);
        GsSwapDispBuff();
        GsSortClear(0x0, 0x0, 0x0,
&WorldOrderingTable[outputBufferIndex]);

        GsDrawOt(&WorldOrderingTable[out-
putBufferIndex]);
        FntPrint("Hello World!\n");
        FntFlush(-1);
        }
        ResetGraph(3);
        return 0;
}
```

aprende a

PROGRAMAR

con nelt yaroze

PlayStation

C: TIPOS DE OPERADORES

La comprensión de las expresiones y, por consiguiente, de los operadores, es algo fundamental en la programación en C. Existen varios tipos distintos de operadores. Hoy estudiaremos los aritméticos y los relacionales. Una expresión es un conjunto de variables, constantes y funciones unidos por operadores. Las variables y las constantes ya las hemos estudiado. Las funciones son programas (generalmente pequeños) que pueden o no devolver un valor. Las expresiones devuelven un dato que, dependiendo del caso, puede ser un valor, una cadena de caracteres, un puntero, etc. Los operadores aritméticos son los más sencillos de entender, ya que representan operaciones aritméticas que se utilizan en todos los aspectos de la vida. En ellos se encuentran la suma, la resta, la multiplicación y la división, representados por los signos +, -, * y / respectivamente. c % b en cambio, devuelve el resto de la división c/b, esto es, c-(c/b). Después están los operadores de incremento y decremento que hacen lo propio con la variable a que van asociadas. Dependiendo de la posición en que esté colocado, la expresión puede tener un significado u otro. Por ejemplo: c = b++ significa que c será igual a b y que, posteriormente, se incrementará en 1 el valor de b. En cambio c = ++b incrementa primero b para después asignar su valor a c. Por último está el signo -, que se utiliza para cambiar el signo de la variable o constante. Aclarar que c = c + b equivale a poner c+ = b, con el consiguiente ahorro de trabajo.

Los operadores relacionales son otro asunto muy distinto, aunque de la misma forma, forman parte de expresiones e, igualmente, devuelven un valor. Sirven para evaluar variables, constantes e, incluso, el valor de retorno de una función. Primero se evalúa toda la expresión y, si la condición es verdadera, entonces se devuelven un valor distinto de 0 (en la tabla hemos puesto 1 por problemas de espacio), mientras que si la condición es falsa, se devuelve 0. Por poner un ejemplo, en el listado de la parte inferior de la página se realizan varias evaluaciones. En la primera se comprueba si hay más manzanas que plátanos, como es falso, el valor devuelto en a es 0. En la segunda, como hay más plátanos que peras, se devuelve un número indeterminado mayor que 0, ya que el resultado es verdadero. Por último se comprueba, con la directiva «if» si hay más peras que manzanas. Como en este caso es falso, se imprime la segunda frase «Hay más manzanas que peras», la que va justo detras de la sentencia «else». Si hubiese sido verdadero, la cadena impresa sería «Hay más peras que manzanas». Este es sólo el comienzo. Todavía nos queda por estudiar varios tipos de operadores más, como los operadores lógicos, de asignación y direccionales. Por el momento es suficiente. Estudiad bien lo que aparece en estas páginas y, a ser posible, el més que viene terminaremos con el apartado que nos ocupara durante estos días.

```
void main(void)
{  int a = 100;
   int b = 30;
   printf("%d ", a + b);
   printf("%d ", a * b);
   printf("%d ", a % b);
   printf("%d ", a - b); }

Al ejecutar este programa la
salida por pantalla es:
130 3000 10 70
```

OPERADORES ARITMETICOS

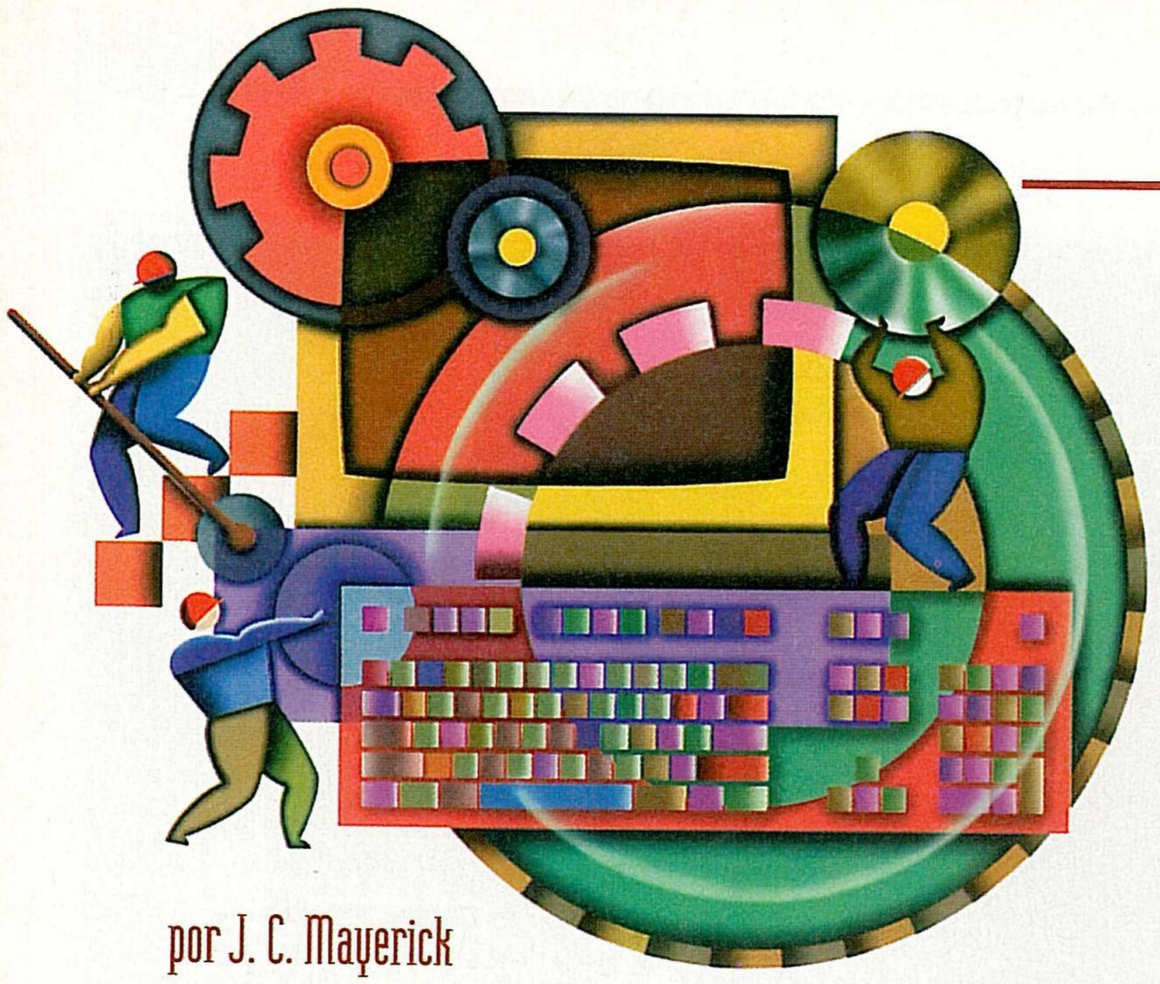
DENOMINACION	OPERADOR	EJEMPLOS
Multiplicación	*	Si a = 2 y b = 3, a*b = 6
División	/	Si a = 9 y b = 3, a/b = 3
Resto de la división	%	Si a = 10 y b = 3, a%b = 1
Suma	+	Si a = 10 y b = 8, a + b = 18
Resta	-	Si a = 6 y b = 2, a - b = 4
Incremento	++	Si a = 7, a++ = 8
Decremento	--	Si a = 6, a-- = 5
Cambio de signo	-	Si a = 13, -a = -13

OPERADORES RELACIONALES

DENOMINACION	OPERADOR	EJEMPLOS
Mayor que	>	Si a = 3 y b = 4, a > b = 0
Mayor o igual que	>=	Si a = 3 y b = 3, a >= b = 1
Menor que	<	Si a = 5 y b = 6, a < b = 1
Menor o igual que	<=	Si a = 7 y b = 7, a <= b = 1
Igual que	==	Si a = 5 y b = 4, a == b = 0
Distinto de	!=	Si a = 5 y b = 6, a != b = 1

```
void main(void)
{  int peras = 10;
   int manzanas = 12;
   int platanos = 13;
   int a,b;
   a = (manzanas > platanos);
   b = (platanos >= peras);
   printf("%d %d", a, b);
   if(peras > manzanas)
      printf("Hay más peras que
manzanas");
   else
      printf("hay más manzanas
que peras");
}
```

CONTROL DEL PAD



por J. C. Mayerick

El mes pasado dejamos la explicación del Listado Tutol.c en su parte más importante, allí donde comenzaba el bucle principal y el uso de las Tablas de Ordenación. Os mostramos, además, los ficheros Pad.h y Pad.c, donde se definen los métodos y valores para el control del pad de PlayStation.

Las cuatro instrucciones siguientes a la llamada a *PadInit* definen la longitud y el origen de las dos tablas de ordenación creadas (0 y 1). La longitud de la tabla se define como una potencia de 2 y puede variar entre 1 y 14. En este caso *OT_LENGTH* se define como 14, siendo la longitud de la tabla de 2^{14}=16384 posiciones. *for(;;)* define el comienzo de un bucle incondicional que se ejecutará infinitamente hasta la aparición de una instrucción *break*. A continuación se lee el estado del *control pad* y se chequea el valor recibido. Si el *bit* 8 está a uno quiere decir que el botón *Select* está siendo pulsado, y por tanto se pone fin a la ejecución del bucle con una instruccion break. Si la condición es falsa, se busca el *buffer* activo en ese momento (0 ó 1), y con dicho valor se selecciona el área de memoria que albergará las primitivas generadas por determinadas instrucciones y que, posteriormente, la *GPU* utilizará para generar la salida por pantalla. Ya sólo queda borrar (inicializar) la tabla de ordenación del *buffer* activo y esperar a que termine de ser presentado en pantalla el otro *buffer*. Antes de volcar el contenido del *buffer* activo es necesario esperar a que se produza una interrupción de sincronismo vertical, para evitar así la aparición de molestas líneas o parpadeos indeseables. *GsSortClear* introduce un comando de borrado de pantalla en la tabla de ordenación, con varios argumentos que indican el color *RGB* a utilizar y la dirección de la tabla del *buffer* activo. *GsDrawOt* ejecuta las primitivas almacenadas en la tabla (en este caso sólo Borrar). Efectivamente, tanto lío con las tablas de ordenación y sólo nos ha servido en este ejemplo para borrar la pantalla. Las dos siguientes instrucciones imprimen la frase «Hello World» mientras que las dos últimas detienen toda actividad gráfica, terminando la ejecución devolviendo el valor 1 de que todo ha ido bien.

```
/******************
 *      Pad.h       *
 *                  *
 *  Copyright (C) *
 *  SCE 1996        *
 ******************/

void PadInit (void);
u_long PadRead (void);

#define PADLup      (1<<12)
#define PADLdown    (1<<14)
#define PADLleft    (1<<15)
#define PADLright   (1<<13)
#define PADRup      (1<< 4)
#define PADRdown    (1<< 6)
#define PADRleft    (1<< 7)
#define PADRright   (1<< 5)
#define PADi        (1<< 9)
#define PADj        (1<<10)
#define PADk        (1<< 8)
#define PADl        (1<< 3)
#define PADm        (1<< 1)
#define PADn        (1<< 2)
#define PADo        (1<< 0)
#define PADh        (1<<11)
#define PADL1       PADn
#define PADL2       PADo
#define PADR1       PADl
#define PADR2       PADm
#define PADstart    PADh
#define PADselect   PADk
```

```
/******************
 *      Pad.c       *
 *                  *
 *  Copyright (C) *
 *  SCE 1996        *
 ******************/

#include <libps.h>
#include "pad.h"

volatile u_char *bb0, *bb1;

void PadInit (void)
{
        GetPadBuf(&bb0, &bb1);
}

u_long PadRead(void)
{
        return(~(*(bb0+3) | *(bb0+2) << 8
        | *(bb1+3) << 16 | *(bb1+2) <<
        24));
}
```

C: TIPOS DE OPERADORES

Continuamos con la explicaciones de los operadores, centrándonos en esta ocasión en los lógicos y de asignación. Los operadores lógicos, como su propio nombre indica, permiten realizar operaciones lógicas con variables, constantes, etc. Debemos diferenciar entre dos grupos distintos: condicionales y de manejo de *bits*. Los primeros entran en juego en la toma de decisiones, bien a través de sentencias *if*, bien a través de bucles *for*, *while*, etc. Así, *A && B* devuelve un valor distinto de cero (verdadero) sólo cuando A y B son verdaderos. *A || B* devuelve verdadero cuando uno u otro son verdaderos (o los dos). Mientras *!B* devuelve falso si B es verdadero o lo contrario si B es falso. Los cuatro operadores lógicos restantes realizan operaciones «Booleanas» (algebra de **Bool**) a nivel de *bit*. *A & B* es un Y lógico que devuelve 1 sólo cuando A = 1 y B = 1. *A | B* (O lógico) devuelve 1 cuando A ó B son iguales a 1. El O exclusivo, en cambio, sólo devuelve 1 cuando A y B son distintos (A = 0 y B = 1 ó A =1 y B = 0). *<< y >>*, por último, realiza desplazamientos a nivel de *bit*, rellenando los espacios sobrantes con ceros. Estos dos últimos operadores resultan especialmente interesantes a la hora de realizar multiplicaciones o divisiones por 2 ó por potencias de 2. Dos ejemplos: 16 << 2 es igual que 16 * 4; mientras que 16 >> 4 es equivalente a 16 / 16. Si observáis un poco, comprobaréis que 2^2 es igual a 4, mientras que 2^4 es igual a 16. Utilizar este método puede ahorrar tiempo de ejecución, ya que existen varios *nmotécnicos* en ensamblador para este tipo de operaciones.

```
void main(void)
{  int a = 16945;
   char b = 20;
   a &= (int) b; }

Suponiendo que int tenga una
longitud de 16 bits:
16945d = 0100001000110001b
   20d = 0000000000010100b
El resultado sería:
   16d = 0000000000010000b
```

OPERADORES LOGICOS

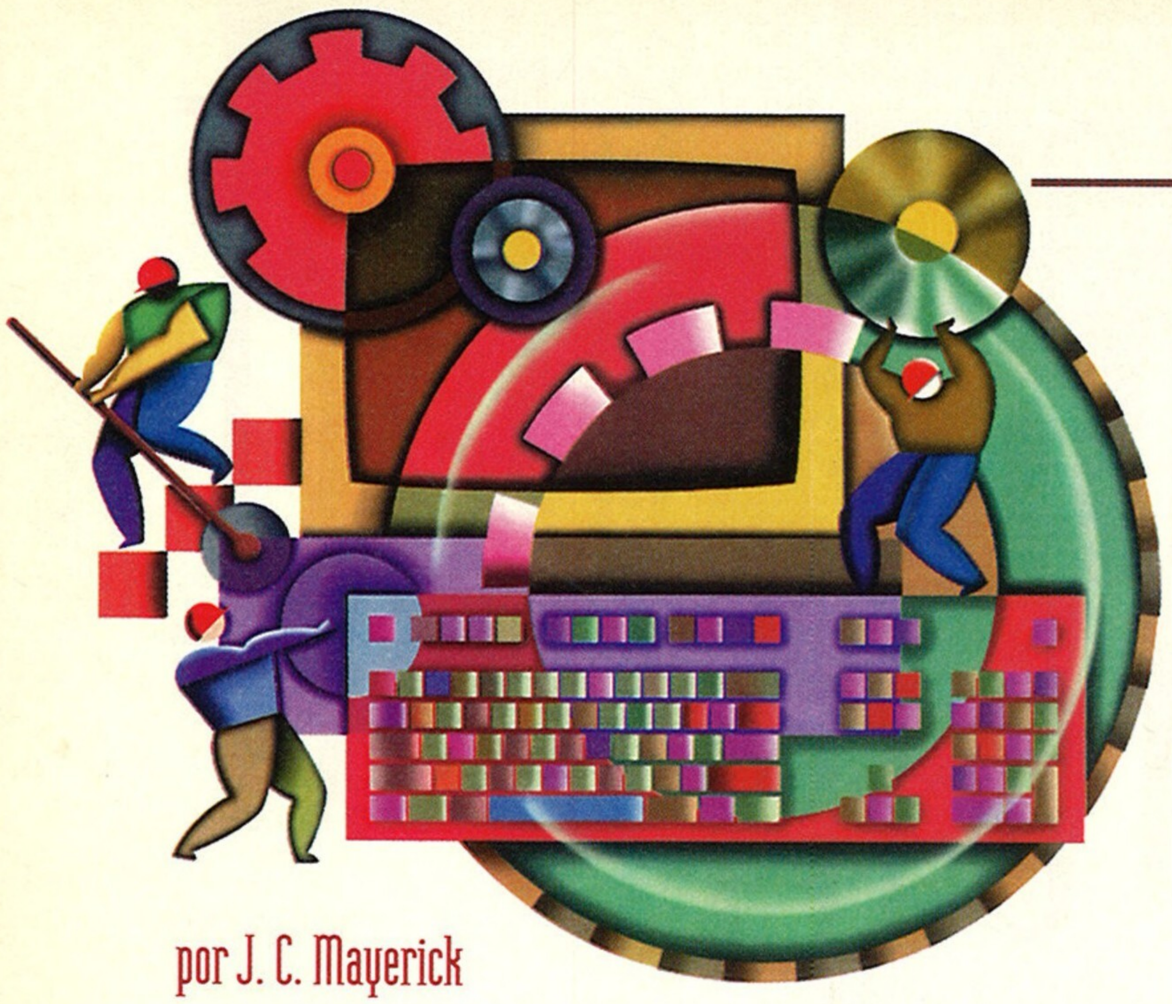
DENOMINACION	OPERADOR	EJEMPLOS
Y lógico (condicional)	&&	año==1997 && mes==8
O lógico (condicional)	||	año==1996 || mes==8
Negación lógica (cond.)	!	!(año==1987)
Y lógico (manejo de bits)	&	0010 & 0110 = 0010b
O lógico (manejo de bits)	|	1001 | 0011 = 1011b
O exclusivo (m. de bits)	^	1011 ^ 0111 = 1100b
Desplazamiento izquierda	<<	01101011 << 2 = 10101100b
Desplazamiento derecha	>>	01101011 >> 2 = 00011010b

OPERADORES DE ASIGNACION

DENOMINACION	OPERADOR	EJEMPLOS
Asignación	x = y	Asigna el valor de y a x
Incremento en una unidad	x++	Incrementa x en una unidad
Decremento en una unidad	x--	Decrementa x en una unidad
Incremento	x += y	Suma a x el valor de y
Decremento	x -= y	Resta a x el valor de y
Multiplicación	x *= y	Multplica el valor de x por y
División	x /= y	Divide el valor de x entre y
Resto de división	x %= y	x igual a resto de x entre y
Desplazamiento izquierda	x <<= 2	Desplaza a x dos bits a Izq.
Desplazamiento derecha	x >>= 2	Desplaza a x dos bits a Der.
Y lógico (manejo de bits)	x &= 0x2f	x igual a x & 0x2f
O lógico (manejo de bits)	x |= 4	x igual a x | 4
O exclusivo (m. de bits)	x ^= y	x igual a x ^ y

Los operadores de asignación, en cambio, no merecen demasiadas explicaciones. Si echáis un vistazo a la tabla de operadores, comprobaréis que todos ellos los hemos estudiado ya, bien en este mes, bien en el anterior. Sólo hay algo que los diferencia, y es que en esta ocasión los estamos utilizando para asignar valores concretos a una variable. Así, el operador de asignación principal, el signo igual, puede variar su comportamiento dependiendo del prefijo que se le incorpore. Un ejemplo: *x* += y asigna a x el valor obtenido tras la suma de x e y. Este ejemplo sirve para el resto de operadores, aunque nuevamente resulta imprescindible realizar un par de diferenciaciones. Por un lado están los operadores de incremento y decremento, que como ya se explicó en el pasado número, incrementan en una unidad el valor de la variable. Por otro, y al igual que ocurre con los lógicos, están los operadores de manejo de *bits*. Con estos hay que tener sumo cuidado, ya que en *A & B*, A debe ser del mismo tipo que B (*char*, *int*, etc.). En caso de que no sea así, se puede recurrir a los *typecast*, que permiten «redondear» una variable o constante a un determinado tipo de dato, extendiendo el signo de éste (el último *bit*). En el ejemplo se comprueba como realizar este tipo de operaciones y el resultado obtenido.

No os calentamos más la cabeza. Estudiad y, a ser posible, practicar.



por J. C. Mayerick

El kit de desarollo de Sony se está abriendo camino a pasos agigantados. Ya en el ECTS pudimos obervar la buena disposición de Sony para potenciar la utilización de sus herramientas de programación entre los más jóvenes. Allí nos encontramos con un CD promocional en el que se habían incluido algunos de los mejores trabajos que los miembros de la WEB han creado. Además

pudimos observar una demo de Final Fantasy hecha con la Yaroze simplemente alucinante (aunque no disponible para el público en general). Este mes os mostramos estas demos de las que os hablamos, a modo de descanso antes de introducirnos de lleno en la programación pura y dura. El mes que viene llegarán los primeros problemas de esta fascinante aventura. Aprenderemos a plantear un proyecto y, lo mejor de todo, plasmarlo en la pantalla de nuestro monitor.

YAROZE DEMO CD

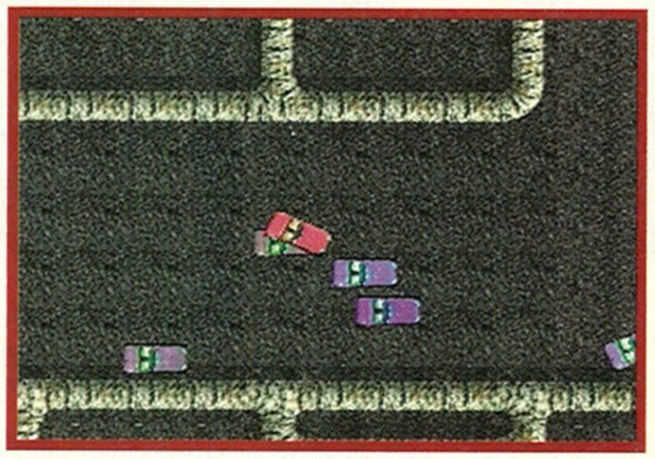
3D TUNNEL

Esta primera *demo* muestra un circuito en 3D al través del cual nos moveremos con una pseudo-nave. Permite además cambiar de resolución, con lo que se aprecian las diferencias existentes entre los distintos modos de pantalla. La suavidad con que el entorno 3D se mueve es una buena muestra de la capacidad 3D de ***Yaroze***. Por lo demás tampoco hay mucho que ver, aunque fijarse en los parámetros que se muestran en pantalla siempre vienen bien.

BREAK 3D

Un perfecto ejemplo en el que se muestra la forma de integrar como textura un fondo 2D dinámico. Es el clásico BREAK-OUT pero con una concepción totalmente innovadora que nos hará perder los nervios en un tiempo muy breve.

CAR G1

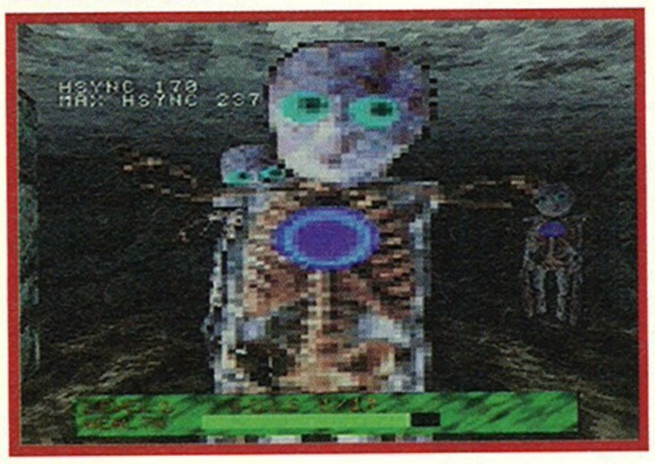


No es ninguna maravilla, carece de detección de choques, pero sirve como excelente muestra de lo sencillo que puede llegar a ser el hacer un juego con ***Yaroze***. Utiliza rotaciones para que el giro de los vehículos sea algo más perfecto.

aprende a PROGRAMAR con net yaroze

PlayStation

CLONE



La sencillez es a veces sinónimo de adicción en estado puro. Un *shoot'em-up* 3D en perspectiva subjetiva muy al estilo de leyendas como WOLFENSTEIN, aunque algo mejor técnicamente. Un clásico ya de quienes frecuentamos la WEB de ***Yaroze***.

SOUND TO LIGHT

Aunque ***PlayStation*** lleva su propio reproductor de CDs de audio, siempre es posible acceder a alguna utilidad *software* que mejore el ya incluido. SOUND TO LIGHT, por ejemplo, es un reproductor algo sobrio, aunque cuenta con la posibilidad de armonizar la música del CD con efectos gráficos que la consola irá generando en tiempo real. No es excesivamente espectacular, aunque cuando menos resultará curioso. No es de lo peor del CD.

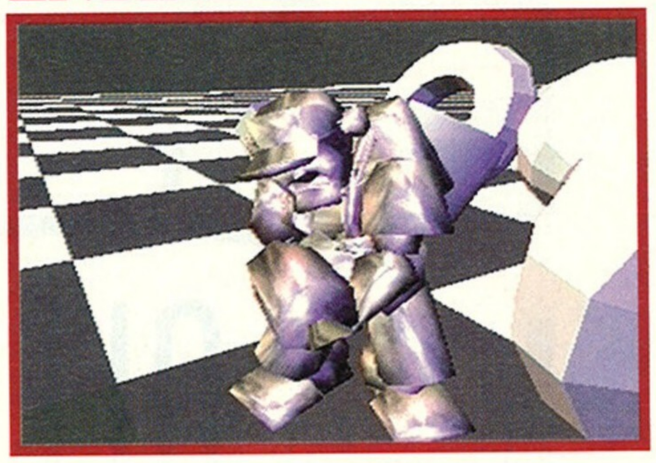
CONEMAN

Uno de los miembros más activos de la WEB (que ya apareció en estas páginas) ha diseñado este PAC-MAN en 3D. El resultado final es bastante aceptable, aunque le falta algo de jugabilidad. Habrá que estar atento a lo próximo de **Coneman**.

FLAG

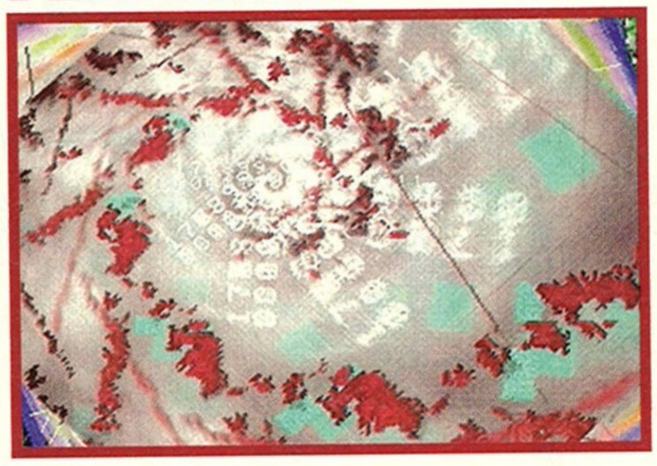
Crear el efecto de una bandera ondeando al viento puede parecer algo excesivamente complicado. Con esta *demo* se podrá observar con detalle los distintos pasos a seguir para lograr semejante efecto. Para ello no habrá más que modificar con la ayuda del *pad* los distintos parámetros que entran en juego. Servirá ádemás como un magnífico ejercicio para aprender a plantear en un futuro el modo de desarrollar efectos gráficos semejantes.

ENGINE 6



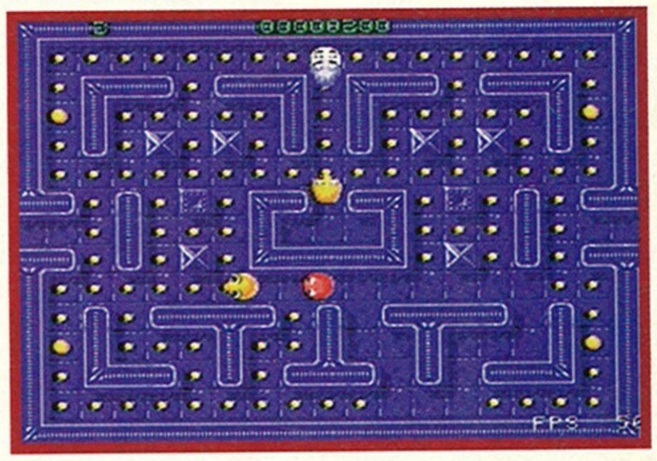
Esta estupenda *demo* pone de relieve la posibilidad de crear materiales complejos en los que la luz juega un papel importante. Un increíble robot metalizado que bien podría haber sido el protagonista de un videojuego comercial.

FEEDBACK



Demo tipo ***Amiga*** de esas que no aportan nada pero que resultan algo vistosillas. Demuestra los interesantes efectos que se logran cuando se entremezclan rotaciones con gráficos y, sobre todo, cuando no se borra el *frame buffer*.

MANIC X

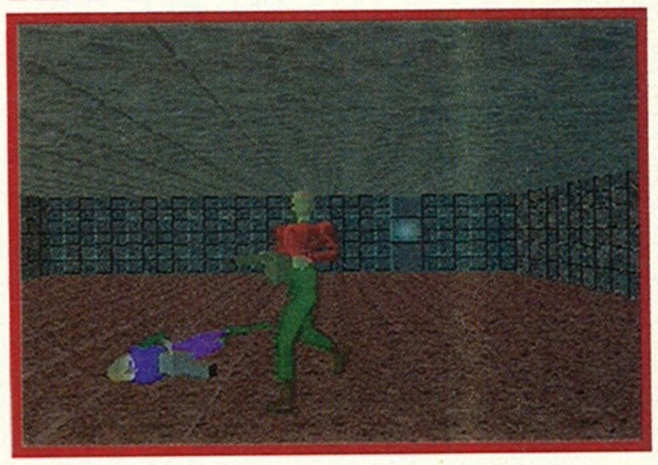


Versión moderna y en 2D del clásico PAC-MAN, aunque con unos gráficos que a muchos recordará al clásico MADMIX GAME. Tiene la pinta de estar acabado, con distintos niveles de juego, por lo que resultará incluso agradable de jugar.

SNAVE

La facilidad de ***PlayStation*** para gestionar todo tipo de entornos en 3D hace que resulte sencillo crear *shoot'em-up* en 3D como este SNAVE. Aunque no esta muy desarrollado, muestra los buenos resultados que una conjunción de *engine 3D* y *sprites 2D* puede proporcionar. De cualquier forma es necesario recordar que, en ciertas ocasiones, un buen polígono puede resultar mucho más efectivo que un *sprite*. Sobre todo en determinados campos.

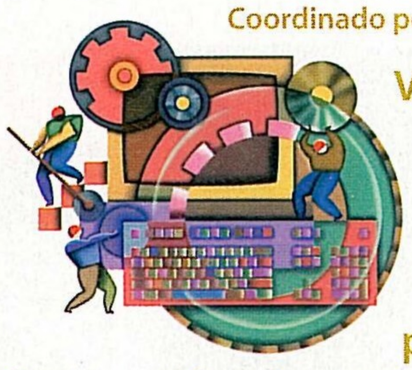
HSFX



La anécdota del CD es este HSFX, una *demo* realmente mala que para nada hablará bien de las posibilidades de la ***Yaroze***. Escenarios pésimos, movimientos muy bruscos y personajes de risa. Un desprestigio para el *kit* de desarrollo de ***Sony***.

aprende a PROGRAMAR con net yaroze

Coordinado por **J.C. MAYERICK**



Volvemos por fin «al tajo» tras unos meses de descanso en los que hemos tratado de replantear la sección (coincidiendo con la nueva imagen de la revista). Hemos prescindido del curso de C porque pensábamos que el progreso era demasiado lento y, además, consumía un espacio precioso. Por eso nos centramos en la programación pura y dura de *PLAYSTATION* (hay muy buenos libros de C por ahí), y lo hacemos con un mini-juego que, a buen seguro, os sonará. El clásico PONG, cuya versión para *PSX* encontraréis en las próximas 155 líneas de código.

MANOS a

Para empezar, nada mejor que PONG, un juego sin complicaciones gráficas que servirá a la perfección para entender la forma de desarrollar un futuro juego. El listado que aparece en la página siguiente se divide en tres partes bien diferenciadas. Declaraciones y definición, bucle principal y funciones de apoyo. Trás la definición de las tablas de ordenación (***NET YAROZE*** nº 64 y nº65), se procede a definir las funciones y las variables que PONG utilizará. Después se crean dos *arrays*. El primero, `pant`, de 14 elementos`RECT`, almacena las coordenadas de las líneas (que en realidad son rectángulos rellenos) que conformarán el escenario. Observad que los dos primeros elementos están inicializados a 0. Esto es así porque en ellos se alojarán posteriormente las coordenadas de las paletas de ambos jugadores. Esto, que puede resultar algo extraño, permitirá más tarde utilizar una única rutina de detección de choques, tanto para el escenario como para las paletas. El segundo *array* almacena los números de los marcadores (cómo están almacenados se explicará más adelante).

Ya en el módulo `Main` inicializamos el entorno de trabajo (ya se explicó en la sección en los números 64 y 65), para después pasar al bucle principal. Empezamos leyendo el *pad* y comprobando los botones pulsados (la información del *pad* del segundo jugador se almacenan en los *bits* 16 a 31 de `PadStatus`). También se comprueba si las paletas han sobrepasado sus límites de movimiento (46 como límite superior y 224 como límite inferior), ajustándolo en el caso de que así sea. De aquí se pasa a la la selección del *buffer* que se utilizará para almacenar las primitivas con la información de los rectángulos que forman la pantalla. Antes de todo, y esto es especialmente delicado, hay que introducir una primitiva de borrado de pantalla, con `GsClearOt`. Acto seguido se llama a `MueveBola`, que hace varias cosas. Primero actualiza `bx` (que es la coordenada de la bola) sumándole `bxx` (que es el número de *pixels* que se moverán). Comprueba después si colisiona con algo, llamando a la función `Colision`, una función muy sencilla (y llena de fallos), pero que resulta corta y efectiva. Seguidamente hace lo mismo con la coordenada «y» de la bola.

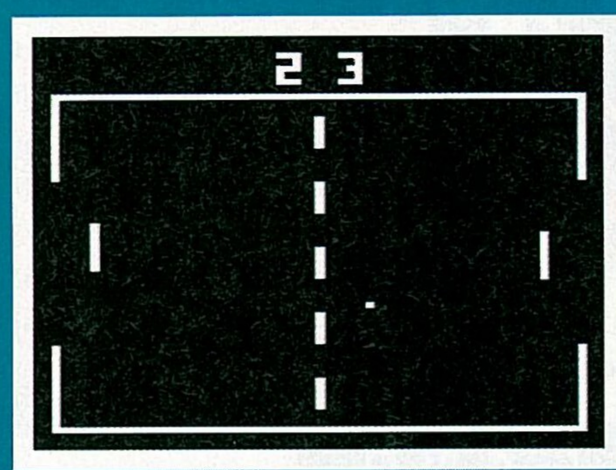
control del tanteo

Lo último que hace `MueveBola` es comprobar, de una forma sencilla, si se ha marcado gol. Para ello mira si la bola sobrepasa las coordenadas 1 y 320 respectivamente para cada jugador. En caso de que así sea, incrementa en 1 las variable `tnt` del jugador correspondiente. Tras regresar de `MueveBola` se llama a `DibPalBol`, que introduce en el *array* `pant` las coordenadas de las paletas (en `pant[0]` y `pant[1]`) y de la bola (en`pant[13]`). Las dos siguientes llamadas a `DibNumeros` permite dibujar el marcador de cada jugador. A esta función se llama especificando el número a dibujar, las coordenadas y el *buffer* de la tabla de ordenación en el que se introducirán las primitivas. El último paso es llamar a `DibPantalla`, que se encargará de introducir el *array* `pant` como primitivas de dibujo de rectángulo (estructuras `GsBOXF`) en la tabla de ordenación del *buffer* activo. Tras esto ya sólo queda esperar a que el barrido de pantalla finalice para llamar a `GsDrawOT`, que se encargará de dibujar, uno por uno, todos los elemento introducidos en la tabla de ordenación. No más por hoy. El próximo mes profundizaremos, con esquemas, en las diferentes funciones que entran en juego en `Pong.c`.

Listado: Pong.c

Este código fuente se completa con el archivo PAD.H publicado en el número 65 de Superjuegos y que se entrega con el Kit de desarrollo de la Net Yaroze.

```
#include <libps.h>
#include "pad.h"

#define PACKETMAX          (10000)
#define PACKETMAX2         (PACKETMAX*24)
#define EIGHT_BIT_MASK     (1 << 24)
#define OT_LENGTH          (14)

GsOT     WorldOrderingTable[2];
GsOT_TAG zSortTable[2][1<<OT_LENGTH];
PACKET   GpuOutputPacket[2][PACKETMAX2];

void DibPantalla(GsOT*
     WorldOrderingTable);
void DibNumeros(int n, int x, int y,
     GsOT* WorldOrderingTable);
void Iniciar(void);
void DibPalBol();
void MueveBola();
```



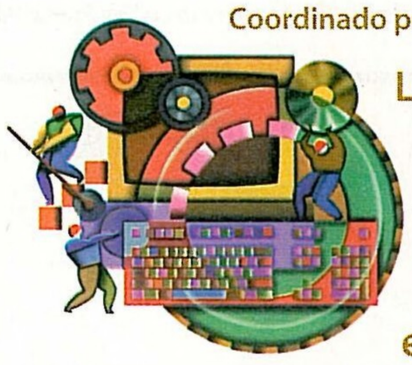
la obra

```
char Colision(void);

int x1, x2, y1, y2, h1, h2;
int bx, by, bxx, byy, tnt1, tnt2;
RECT pant[14] = { {0,0,0,0}, {0,0,0,0},
     {20,30,280,4}, {20,234,280,4},
     {20,34,4,50}, {296,34,4,50},
     {20,184,4,50}, {296,184,4,50},
     {158,44,4,20}, {158,84,4,20},
     {158,124,4,20}, {158,164,4,20},
     {158,204,4,20}, {0,0,4,4} };
unsigned short numeros[10]={ 0x7b6f,
          0x3249, 0x73e7, 0x72cf,
          0x5bc9, 0x79cf, 0x79ef,
          0x72c9, 0x7bef, 0x7bc9};

int main (void)
{    RECT rect; u_long PadStatus = 0;
     int outputBufferIndex;
     Iniciar();
//   SetVideoMode( MODE_NTSC );
     SetVideoMode( MODE_PAL );
     ResetGraph(0);
     GsInitGraph(320 ,256,
       GsOFSGPU|GsNONINTER, 1, 0);
     GsDefDispBuff(0, 0, 0, 256);
     FntLoad(960, 256);
     FntOpen(32, 32, 256, 200, 0, 512);
     PadInit();
     WorldOrderingTable[0].length =
       OT_LENGTH;
     WorldOrderingTable[1].length =
       OT_LENGTH;
     WorldOrderingTable[0].org =
       zSortTable[0];
     WorldOrderingTable[1].org =
       zSortTable[1];
     for (;;)
     {    PadStatus = PadRead();
          if(PadStatus & PADselect) break;
          if((PadStatus & PADLup) &&
             (y1>=44)) y1-=2;
          if((PadStatus & PADLdown) &&
             (y1<=224-h1)) y1+=2;
          PadStatus = PadStatus >> 16;
          if((PadStatus & PADLup) &&
             (y2>=44)) y2-=2;
          if((PadStatus & PADLdown) &&
             (y2<=224-h1)) y2+=2;
          outputBufferIndex =
             GsGetActiveBuff();
          GsSetWorkBase((PACKET*)GpuOutput
Packet[outputBufferIndex]);
          GsClearOt(0, 0, &WorldOrderingTa
ble[outputBufferIndex]);
          MueveBola();
          DibPalBol();
          DibNumeros(tnt1, 138, 5, &WorldO
rderingTable[outputBufferIndex]);
          DibNumeros(tnt2, 170, 5, &WorldO
rderingTable[outputBufferIndex]);
          DibPantalla(&WorldOrderingTable[
outputBufferIndex]);
          VSync(0);
          ResetGraph(1);
          GsSwapDispBuff();
          GsSortClear(0, 0, 0, &WorldOrder
ingTable[outputBufferIndex]);
          GsDrawOt(&WorldOrderingTable[out
putBufferIndex]);
          FntFlush(-1);
     }
     ResetGraph(3); return 0;
}

void Iniciar(void)
{         x1=40; x2=276; y1=115; y2=115;
          h1=30; h2=30; bx=158; by=116;
          bxx=-2; byy=-2; tnt1=0; tnt2=0;
}

void DibPantalla(GsOT*
     WorldOrderingTable)
{    GsBOXF linea;
     int i; linea.attribute=0;
     linea.r=255; linea.g=255;
     linea.b=255;
     for(i=0;i<14;i++)
     {    linea.x=pant[i].x;
          linea.y=pant[i].y;
          linea.w=pant[i].w;
          linea.h=pant[i].h;
          GsSortBoxFill(&linea,
            WorldOrderingTable, 0); };
}

void DibPalBol()
{    pant[0].x=x1; pant[0].y=y1;
     pant[0].w=4; pant[0].h=h1;
     pant[1].x=x2; pant[1].y=y2;
     pant[1].w=4; pant[1].h=h2;
     pant[13].x=bx; pant[13].y=by;
}

void MueveBola(void)
{  int i=0, flagx=0, flagy=0, fin=1;
   bx+=bxx;
   if(Colision() == 1)
   {    bxx*=-1; bx+=bxx; }
   by+=byy;
   if(Colision() == 1)
   {    byy*=-1; by+=byy; }
   if(bx<=1) {tnt1++;bx=158;bxx*=-1;}
   if(bx>=320) {tnt2++;bx=158;bxx*=-1;}
}

char Colision(void)
{    int i; char flag=0;
     for(i=0; i<8; i++)
     { if(bx >= (pant[i].x-4) &&
          bx <= (pant[i].x+pant[i].w) &&
          by >= (pant[i].y-4) &&
          by <= (pant[i].y+pant[i].h))
          flag=1; }
     return flag;
}

void DibNumeros(int n, int x, int y,
     GsOT* WorldOrderingTable)
{    GsBOXF linea;
     int i,j; linea.attribute=0;
     linea.r=255; linea.g=255;
     linea.b=255; linea.w=4;
     linea.h=4; n=numeros[n];
     for(i=4; i>=0; i--)
          for(j=2; j>=0; j--)
          {    if(n & 1)
               {    linea.x=x+(4*j);
                    linea.y=y+(4*i);
                    GsSortBoxFill(&linea,Wor
ldOrderingTable, 0); }
          n=n>>1;
          }
}
```

Comprender la forma en que trabaja el SDK de Net Yaroze resulta especialmente importante.

aprende a programar con net yaroze

Coordinado por **J. C. MAYERICK**



Lentos pero seguros. Esa es la máxima de esta sección, muy acorde a las capacidades psíquicas de su coordinador. Hoy culminamos nuestro primer trabajo, PONG.C, un experimento muy sencillo pero que, espero, os haya abierto a todos las puertas de la programación en *PLAYSTATION*. En los próximos meses entraremos en cosas mucho más complicadas, como *sprites*, fondos y demás, aunque mientras, como aperitivo, os presentamos un título programado con el *KIT NET YAROZE* que, como se puede comprobar, cuenta con cierto nivel de calidad.

El mes pasado publicamos el listado de nuestro primer «juego» `Pong.c`. En esta segunda entrega os mostramos el *modus operandi* de las principales funciones que entran en juego en este programa excepto la forma en que trabaja la función principal *Main*, ya que se explicó su funcionamiento en números anteriores. El mes que viene avanzaremos más e intentaremos cosas más complicadas.

INICIALIZACION

- **FUNCION: iniciar**
- **ENTRADA: nada**
- **SALIDA: nada**

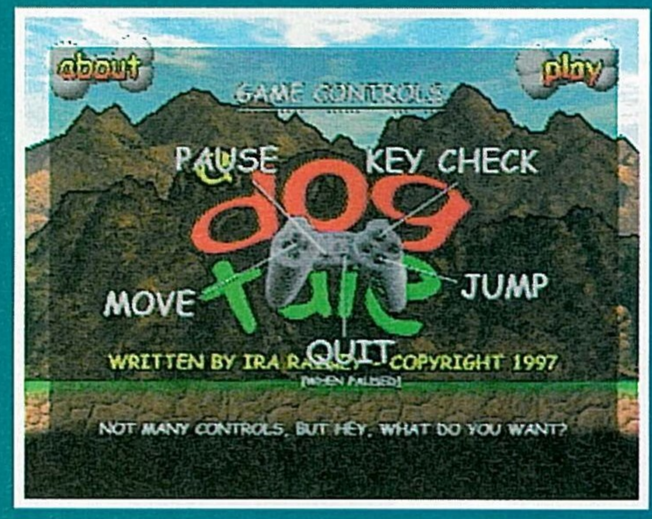
En esta primera función se inicializan las principales variables que van a entrar en juego en `Pong.c`. `x1`, `x2`, `y2` e `y2` representan los valores `x` e `y` para los jugadores uno y dos respectivamente. Mientras `h1` y `h2` representan el largo de las paletas (esto permitiría cambiar su tamaño después), al tiempo que `bx` y `by` almacenan las coordenadas de la bola. `bxx`, `byy`, `tnt1` y `tnt2` guardarán el desplazamiento `x` e `y` de la bola y los tanteos para ambos jugadores respectivamente.

DIBUJAR LA PANTALLA

- **FUNCION: dibpantalla**
- **ENTRADA: nada**
- **SALIDA: nada**

La forma en que se dibuja la pantalla es relativamente sencilla. En el *array* `pant`, compuesto por catorce elementos `RECT` (una estructura `RECT` consta de valores `x`, `y`, `h` y `w`) se introdujo las coordenadas `x` e `y` de cada línea de la pantalla, así como el ancho de cada una de ellas. Los dos primeros elementos del *array* están a cero, ya que en ellos se almacenará posteriormente la

LAS FUN



información referente a las paletas (esto permite utilizar una única rutina de detección de choques para todos los elementos de pantalla). El último, el 13, también se inicializa con los valores `x` e `y` a cero, ya que éstos representan las coordenadas de la bola. El proceso es el siguiente: se crea una estructura `GsBOXF` llamada `linea` en la que se irán almacenando los datos de cada línea de la pantalla. A través de un bucle `for`, que lee los catorce elementos del *array* `pant`, se van introduciendo en la tabla de ordenación por medio de la función `GsSortBoxFill`, que es la encargada de introducir cajas rellenas en nuestra tabla `WorldOrderingTable`. El primer parámetro que esta función requiere es un puntero a una estructura `GsBOXF` (por eso derreferenciamos anteponiendo `&`). En el segundo se pasa la tabla de ordenación en que se introducirá, mientras que el tercero se deja a cero. Por el momento no nos interesa su funcionamiento.

PALETAS Y BOLA

- **FUNCION: DibPalBol**
- **ENTRADA: nada**
- **SALIDA: nada**

`DibPalBol` se encarga de introducir en el *array* `pant` la información concerniente a las paletas y la bola para su posterior dibujado. Los elementos 0 y 1 corresponden a las paletas de ambos jugadores, mientras que el 13 corresponde a la bola.

MOVER LA BOLA

- **FUNCION: MueveBola**
- **ENTRADA: nada**
- **SALIDA: nada**

Esta función actúa, mano a mano, con la que le sigue: `Colision`. Ya que debe calcular la nueva posición de la bola y primero debe conocer si el valor de las variables de desplazamiento `bxx` y `byy` son válidas. Olvidaros de las variables `flagx`, `flagy` y `fin` que se declaran al principio de la función, ya que están ahí para su uso posterior, y por el momento no tienen utilidad. Primero se chequea el desplazamiento `x` de la bola. Se llama a `Colision` y, si el resultado es 0, quiere decir que el desplazamiento es válido. Si no lo fuese, se multiplicaría por -1 la variable de desplazamiento `bxx`, para invertir de esta forma la trayectoria de la bola. Con el desplazamiento `y` se sigue exactamente el mismo proceso.

CONTROL DE COLISIONES

- **FUNCION: colision**
- **ENTRADA: nada**
- **SALIDA: int flag**

La rutina de detección de choque es muy sencilla pero efectiva (tiene fallos pero mínimos). Lo que hace la rutina es comprobar si la bola colisiona con alguno de los ocho primeros elementos del *array* `pant` (los otros cuatro son las líneas centrales de la pantalla). El único procedimiento que realiza es comprobar si, en algún momento, la bola esta justo encima de algún elemento sólido. Si así es, devuelve 1 en `flag` para que luego `MueveBola` se encarga de invertir la trayectoria.

DIBUJO DE TANTEOS

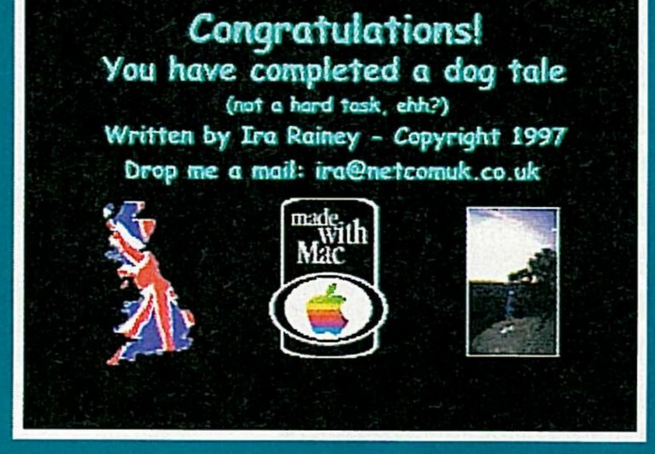
- **FUNCION: DibNumeros**
- **ENTRADA: int, int, int, GSOT***
- **SALIDA: nada**

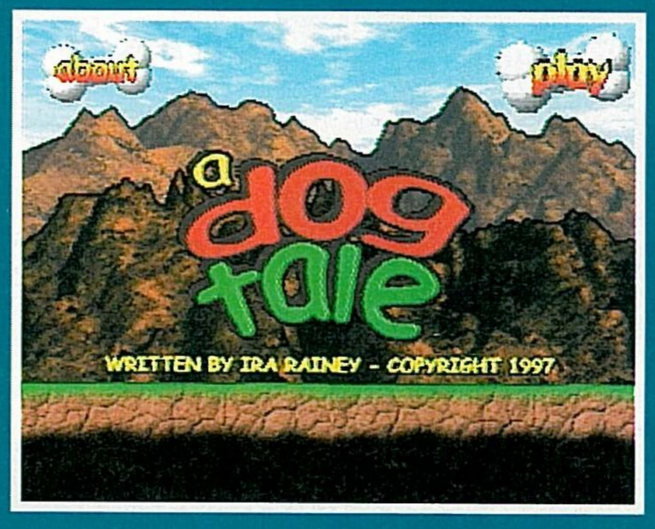
Los números que aparecen en pantalla se almacenan en el *array* `numeros`. Pero ¿cómo? Coged por ejemplo el primer elemento `0x7b6f` y pasadlo a binario, `0111101101101111b`. Eliminad el primer cero y colocar los ceros y los unos en una cuadrícula de 3x3. Comprobaréis que os sale un 0. La rutina `DibNumeros` recoge, *bit* a *bit*, la información correspondiente al número que se quiere imprimir, inroduciendo, si fuera necesario, un cuadro relleno de 4x4 *pixels* en la tabla de ordenación por cada uno de los nueve bloques que componen cada número.

NCIONES

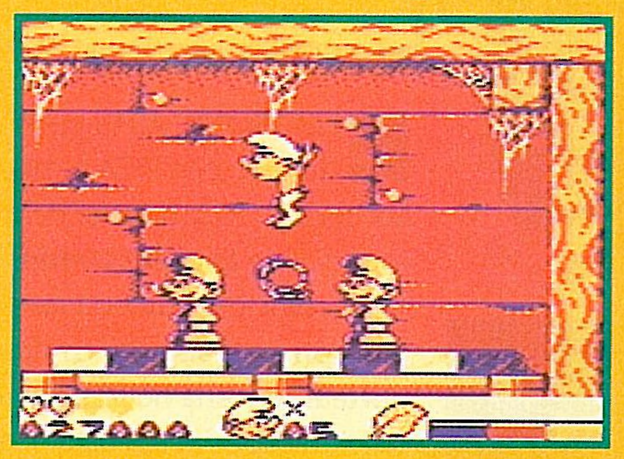
A Dog Tale, en tiempo record

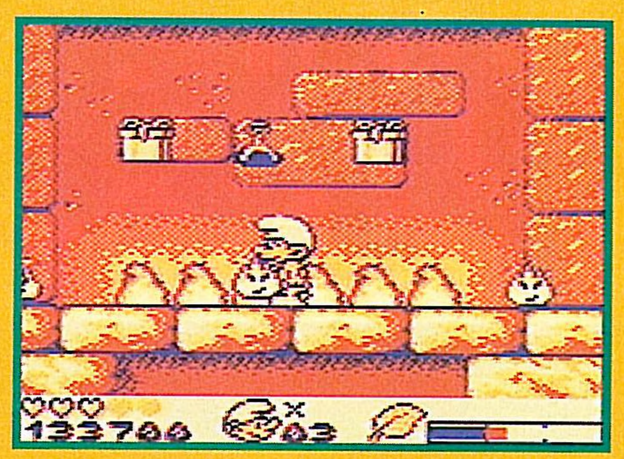
El juego que aparece en este recuadro es el trabajo de un joven británico que lo programó con la ayuda del *KIT NET YAROZE*, en apenas ocho semanas. **A DOG TALE** cuenta con un único nivel, aunque de una dificultad extrema, en el que se deben recoger un número determinado de llaves. Además, y al igual que en juegos tipo DONKEY KONG COUNTRY, hay que recoger una serie de objetos (en este caso huesos) que nos darán vidas extra. Tiene muchos fallos, como es lógico, principalmente en las rutinas de detección de choques, así como la del salto del personaje, un poco «estático» y muy difícil de controlar. Hablando del personaje, destacar el enorme parecido que guarda con nuestro entrañable y «gafotoso» compañero DE LUCAR. Si no fuese por su escaso intelecto y nivel de conocimientos, pensaríamos que él mismo lo había programado. Como curiosidad, está bien.

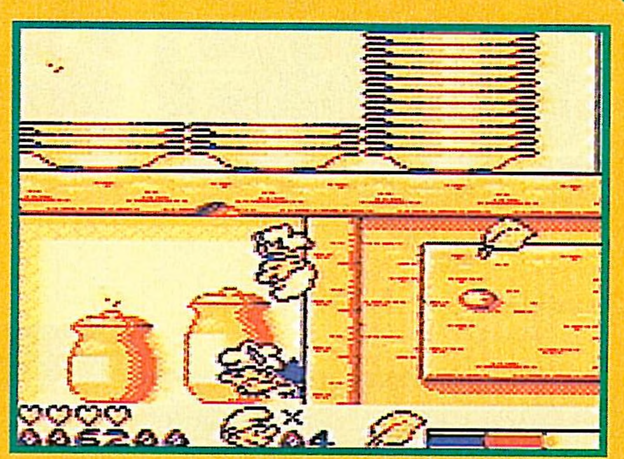


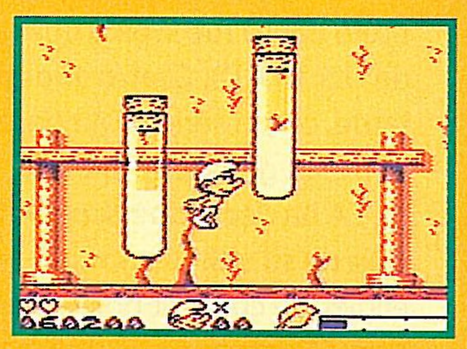


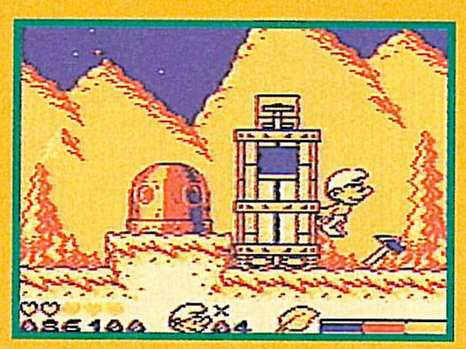
Variedad por encima de todo

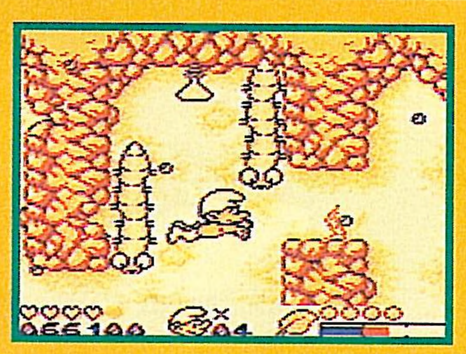




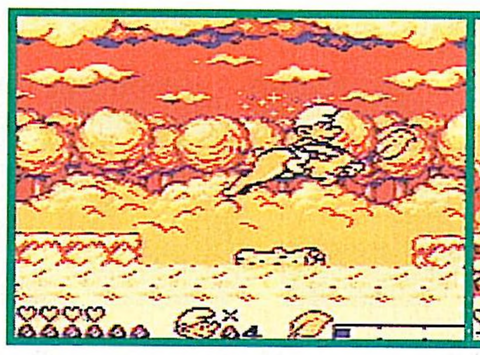


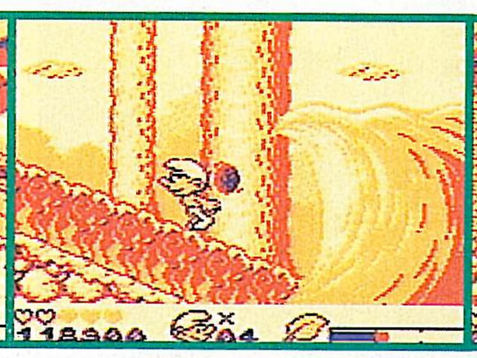


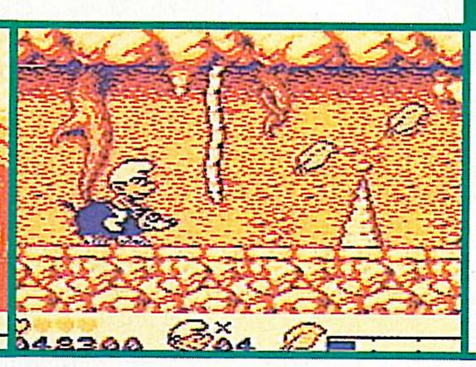




Llegar al final de cada uno de los niveles no será posible si antes no se recoge el número necesario de hojas de zarzaparrilla. Estas le proporcionan a pitufo fortachón la capacidad de realizar saltos mucho mayores y, por consiguiente, de acceder a las zonas más importantes del nivel.





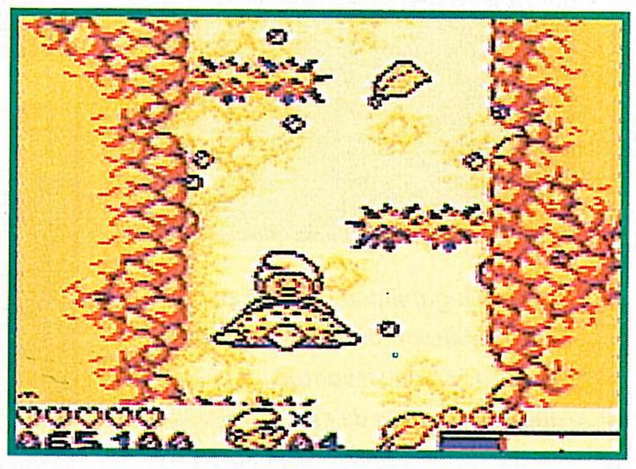


Niveles especiales

Este tipo de niveles se caracteriza por poner a prueba los reflejos del jugador, que en muchas ocasiones no hace más que observar con perplejidad la excelente calidad gráfica y técnica de los mismos. Tanto *scroll parallax* junto no se veía en *Game Boy* desde los tiempos de Speedy Gonzales.

perfección. Igual de sorprendente resulta la fase en la que una ola gigantesca persigue al protagonista. Sabemos de buena fe la dificultad técnica que un efecto así entraña, algo que no parece haber sido obstáculo para las privilegiadas mentes de sus creadores. En el apartado musical, **Los Pitufos 3** nos ha sorprendido gratamente. A la conocida melodía de los pitufos hay que sumar varias más de contenido bastante pegadizo, todas ellas dotadas de notables efectos de percusión. El desarrollo, ante tal variedad, (existe incluso un nivel de *bonus*) ha ganado en diversión, aunque aún podían haberse mejorado las fases centrales del juego. Aún así creemos que **Los Pitufos 3** puede marcar una época a nivel técnico. ¿Os imagináis un nuevo Mario o DKL con *scroll Parallax*? Por soñar que no quede, aunque de momento nos conformamos.

J. C. MAYERICK

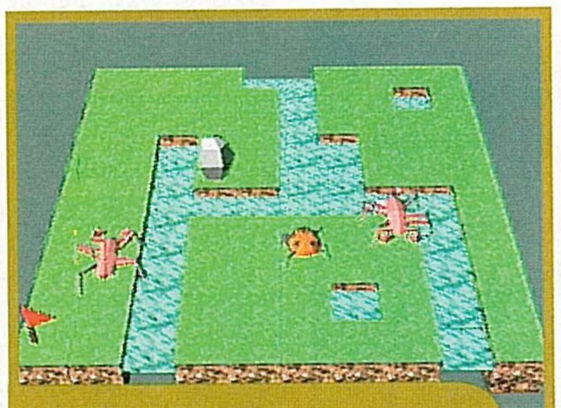


GRAFICOS	MUSICA	SONIDO FX	JUGABILIDAD	GLOBAL
91	92	80	90	90
En este apartado debemos diferenciar dos sub-apartados. Por un lado los gráficos, muy buenos y resultones. Por otro, los efectos gráficos, muy sorprendentes y prácticamente inéditos en una consola como Game Boy.	Es otro de los apartados que te sorprenden por su calidad. La gran calidad de ejecución de sus melodías se ve aderezada con partituras tremendamente pegadizas y llenas de ritmo. cosa extraña en Game Boy.	No se le pueden pedir peras al olmo, igual no se le puede exigir a un juego que sea sobresaliente en todos los apartados. El sonido es lo peor de Los Pitufos 3, aunque teniendo en cuenta todo lo demás, está perdonado.	La primera parte de la saga era divertida. La segunda era terriblemente aburrida. Esta tercera es como la primera pero con las innovaciones necesarias para ser aún más divertida. Infogrames, en esta ocasión, ha acertado.	+ Por fin un juego de Los Pitufos da la talla. - El dichoso sistema de passwords con caretos de los pitufos no puede ser menos práctico.

aprende a programar con net yaroze

Coordinado por **J. C. MAYERICK**

Sorprendidos nos quedamos todos en la redacción cuando tuvimos la oportunidad de acceder al CD de demostración que *Net Yaroze Japon* ha puesto a disposición de la prensa especializada. Aunque las comparaciones sean odiosas, las diferencias entre las *demos* europeas y japonesas son abismales, con especial atención para un grupo de programación que promete mucho: TEAM FATAL. Si hacen esto con *Net Yaroze*...



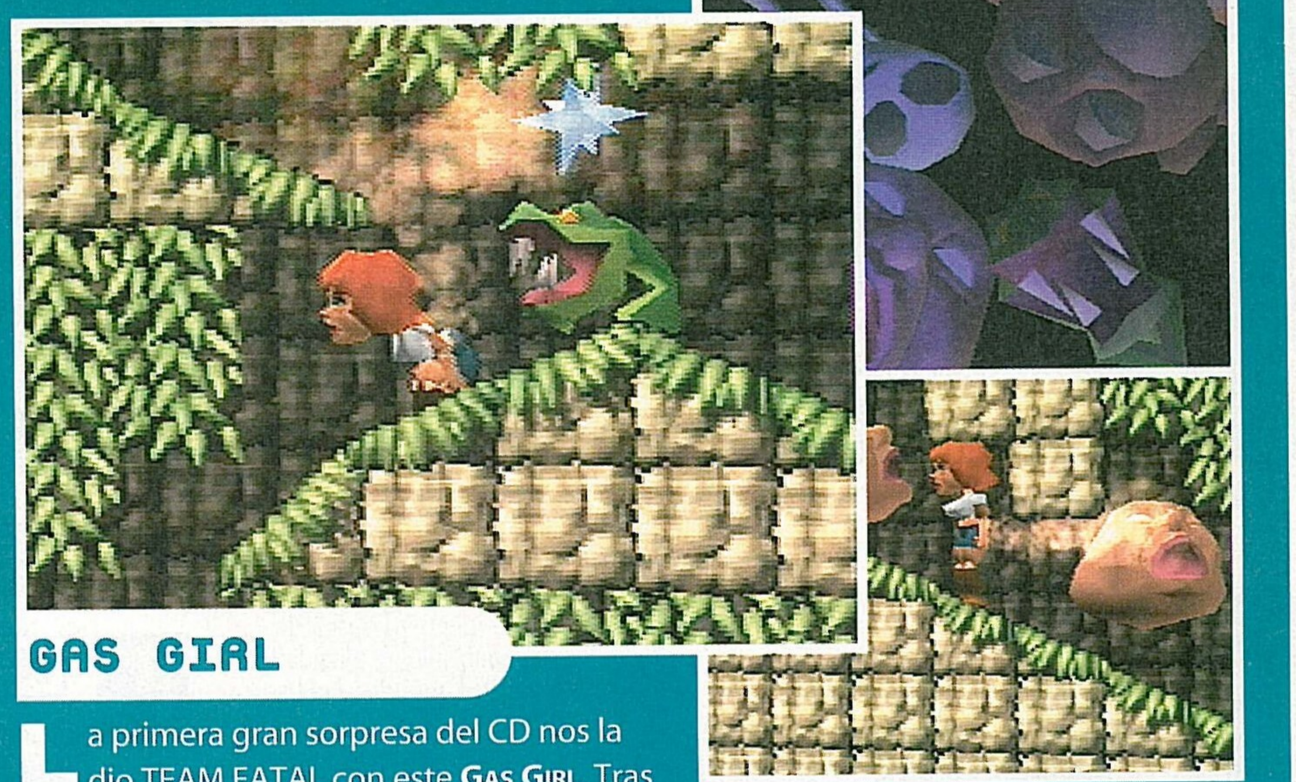
INSECTS

Aunque gráficamente es muy limitado, **Insects** cuenta con la gran facultad de ser especialmente divertido, como corresponde a un sistema de juego que ha sido ya múltiples veces utilizado. En la piel de una mariquita (no como **De Lucar**, sino de las que tienen topitos negros y vuelan), nuestra misión consistirá en alcanzar la bandera que hay en cada uno de los niveles. Para ello habrá que utilizar los *items* que hay sobre el «tablero», lo que permitirá evitar el casi estúpido patrón de ataque de nuestros enemigos. **Insects** es de esos juegos que, a lo tonto, te acaban enganchando durante largos periodos de tiempo.

SHOOT'EM-UP 1

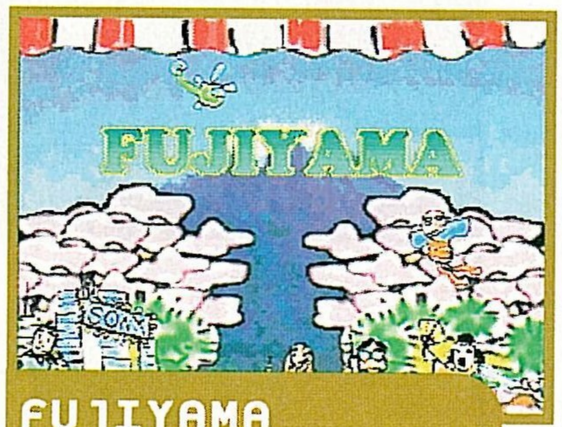
Es de las *demos* más veteranas que se pueden encontrar en la red, y no por ello deja de sorprendernos. A pesar de su exagerada dificultad (en unos segundos habremos muerto), es sorprendente comprobar los estupendos juegos de luces que se pueden llevar a cabo con ***Net Yaroze***. Cuenta además con una virtud a la que todos los aficcionados a los *shoot'em-up* nos aferramos: la pantalla se encuentra continuamente invadida por enemigos, explosiones, rayos y todo tipo de elementos que convierten a este «matamarcianos» en todo un espectáculo visual. Dista mucho de estar finalizado, de hecho hace bastante que su creador no saca nuevas versiones, pero a buen seguro servirá como obra didáctica para aquellos que hagan sus primeros pinitos con ***Net Yaroze***. Además, y si has bajado la *demo* desde **Internet**, podrás editar fácilmente tus propios niveles con un simple editor de texto y un pelín de ingenio.

JAPON SIEMPR



GAS GIRL

La primera gran sorpresa del CD nos la dio TEAM FATAL con este **Gas Girl**. Tras una *Intro* corta pero espeluznante se nos da paso a un juego de plataformas en 3D en la que la protagonista tiene una cierta tendencia a gasear a todo lo que se interpone en su camino. Con dichas ventosidades no sólo asfixia a sus enemigos (como se muestra en las pantallas), sino que además será capaz de tomar un impulso extra para alcanzar ciertas plataformas. El nivel de gases se aumenta recogiendo corazones (unas «fabes» habrían sido más apropiadas). En el apartado técnico, sobresaliente, con un *engine 3D* que destaca por su suavidad y sus geniales efectos gráficos. Sólo se echa en falta un poquillo más de variedad.



FUJIYAMA

Un extraño invento que, al estar en japonés, cuesta bastante interpretar. A primera vista parece un catálogo de montañas y volcanes algo sencillote, aunque si lo hemos acercado hasta estas páginas es por lo curiosa que resulta su pantalla de presentación (la que aparece sobre estas líneas). Una especie de fondo pintado a mano con personajes que, a modo de recortables, se mueven por la pantalla. Un efecto muy curioso que merece la pena observar.

SHOOT'EM-UP 2

A este segundo *shoot'em-up* le ocurre algo parecido al primero que hemos comentado. Cuenta con un único nivel continuo sobre el que aparecen infinidad de enemigos a una velocidad endiablada. En los pocos segundos que dura la partida se puede apreciar una apariencia muy similar a la de los *shoot'em-up* comerciales, lo que da una buena muestra de las capacidad del *SDK* de ***Net Yaroze*** aplicados a los entornos 2D. Esperemos que haya nuevas versiones.

HOVER RACING

Si recordáis el inolvidable F-Zero de ***Super Nintendo***, **Hover Racing** es lo más parecido que se puede encontrar para ***PlayStation***, y lo que es mejor, técnicamente es muy bueno. Es de los pocos juegos que aparecen en el CD que están completamente terminados, ya que cuenta con selección de vehículos, distintos modos de juegos (*Battle*, *Versus*, *Time Attack*, etc.) y nueve circuitos con trazados realmente asombrosos. Incluye un modo de dos jugadores a pantalla partida, aunque a costa de ralentizar un pelín el desarrollo del juego. ¿Lo más sorprendente de **Hover Racing**? La suavidad de movimientos en el modo de un jugador y, lo que es más importante, la increíble sensación de velocidad.

RE EN CABEZA

TERRA INCOGNITA

Terra Incognita es, hoy por hoy, la obra maestra de un equipo de programación que hace sus primeros pinitos con ***Net Yaroze***: TEAM FATAL. A pesar de contar con la apariencia de un *Action RPG*, **Terra Incognita** es la mezcla perfecta entre el desarrollo de LandStalker y la apariencia gráfica de Dark Savior. Su magnífico *engine*, que nos proporciona una libertad total en la selección de vistas, tan sólo flaquea por la ausencia de fondos, que en este caso podría haberse solventado con simples *bitmaps*. La suavidad de movimientos, su magnífica *Intro* y notables músicas hacen que **Terra Incognita** pudiste pasar incluso como un juego comercial. Además tiene los textos en japonés e ¡inglés!

aprende a programar con net yaroze

Coordinado por **J. C. MAYERICK**

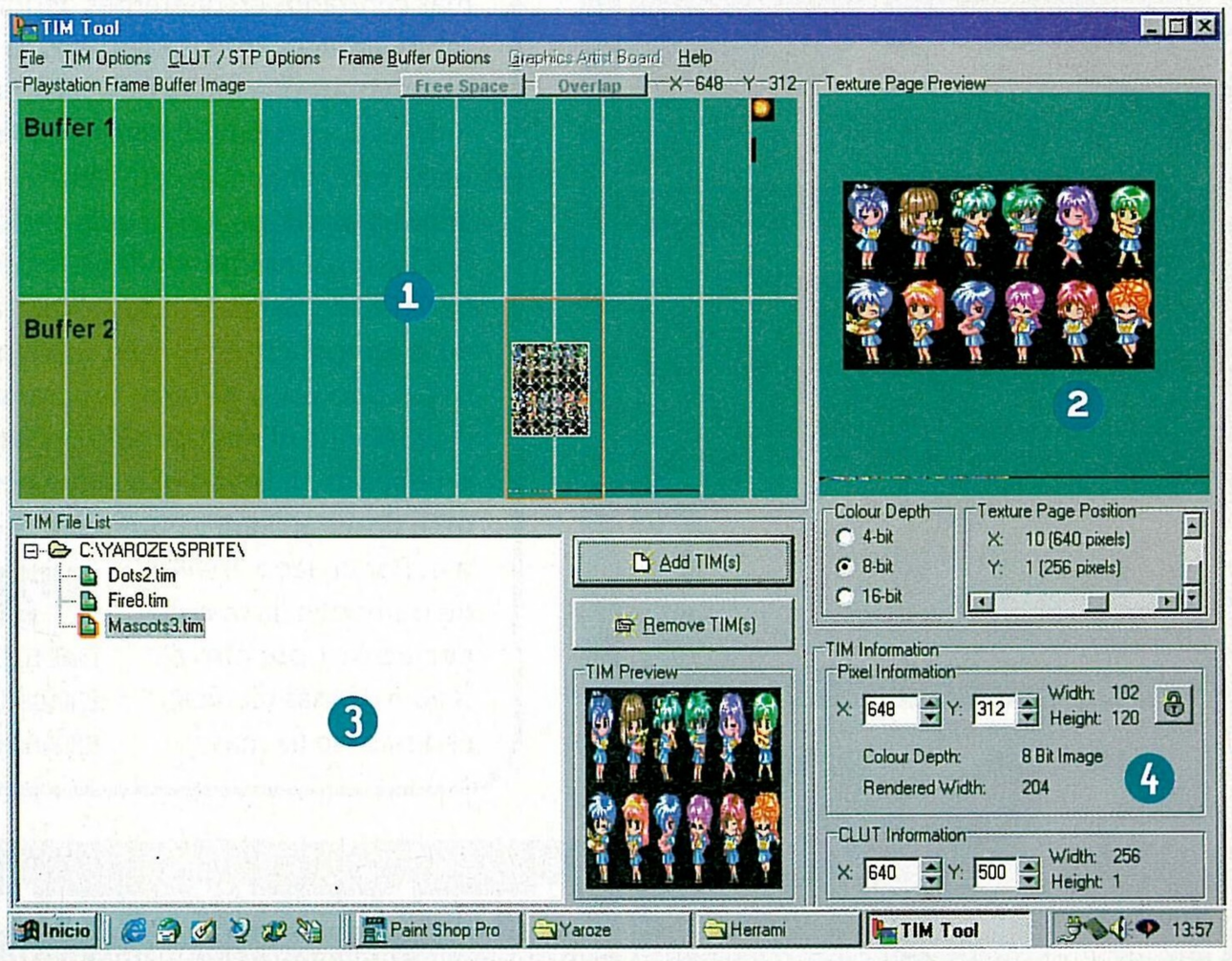
Trabajar con *sprites* sobre *Net Yaroze* es una de las tareas más sencillas que se pueden llevar a cabo, aunque antes de entrar de lleno en el manejo de éstos, es necesario entender cómo trabajar y gestionar convenientemente la VRAM. En este número explicaremos la forma de mover gráficos entre la RAM y la VRAM, así como las diferentes estructuras y funciones que entrarán en juego. De paso, echaremos un vistazo a *TIM Tool*, una excelente herramienta, disponible en la página WEB de *Net Yaroze*, que permite la organización de gráficos y CLUTs sobre la VRAM.

A la hora de comenzar a programar con ***Net Yaroze***, la primera duda que asalta es cómo se mueven los gráficos desde la *RAM* a la *VRAM* (desde este momento, *VRAM* y *Frame Buffer* se refieren a lo mismo). La clave está en la cabecera de los ficheros *TIM*, el formato gráfico utilizado por el *SDK* de ***Yaroze***. En ella se almacena la información concerniente al gráfico, como el modo de *pixel* utilizado, la posición del gráfico y el *CLUT* dentro del *Frame Buffer*, así como punteros a los datos de ambos. Aunque en un principio seremos nosotros los responsables de asignar una dirección de memoria para el almacenamiento del fichero *TIM* en la *RAM*, antes será necesario buscar una posición adecuada para el gráfico y su *CLUT* dentro

CONTROL

Aprovecha al límite el Frame Buffer

El entorno de trabajo de *TIM Tool* se divide en cuatro partes bien diferenciadas, que se explican a continuación: 1) Representación gráfica del *Frame Buffer* o VRAM. En él se observan las diferentes páginas y los dos *Buffer* (en caso de que se utilice *Double Buffering*). 2) Presentación de la página actualmente seleccionada. En la parte inferior se observa una línea con la información del CLUT del fichero *TIM* que aparece. 3) Localización de los diferentes ficheros *TIM* que se han añadido al proyecto. 4) Información concerniente al fichero *TIM* seleccionado y su *CLUT*. Posición, profundidad de color, posición dentro del *Frame Buffer* y de la página, etc. Al guardar el proyecto, *TIM Tool* actualiza las cabeceras de todos los ficheros *TIM*, al tiempo que crea un fichero de texto con toda la información relativa a cada uno de los ficheros *TIM*. Esto último resultará de gran utilidad.



TimTool te ayudará a ordenar la VRAM

DE SPRITES (I)

del *Frame Buffer*. Esto se puede hacer de una forma artesanal con *TIMUtil* (incluido en el *SDK*) o de una forma mucho más intuitiva con *TIMTool*, que puede descargarse de la sección utilidades de la página *WEB* de **NET YAROZE** (ver recuadro). De cualquier modo, es necesario entender que esta utilidad es esencial en el desarrollo de programas no demasiado complejos a nivel gráfico, en los que estos últimos pueden ser acomodados sin apreturas dentro del *Frame Buffer*. Para aplicaciones más complejas, habrá que desarrollar sistemas de gestión de la *VRAM* mucho más específicas, algo que sólo se logra con mucha experiencia. De vuelta a lo que nos ocupa, supongamos que cargamos un fichero *TIM* en la posición xxxxxxxxH de la *RAM*. Para poder mover este gráfico al *Frame Buffer*, primero necesitamos obtener distinta información contenida en la cabecera del fichero. A tal efecto creamos una estructura `GsIMAGE`, con la que después llamaremos a la función `GsGetTimInfo`. A esta función se le deben pasar dos parámetros. Uno, un puntero al gráfico, en este caso xxxxxxxxH, y dos, un puntero a la estructura `GsIMAGE`. Ya que la función nos obliga a que este último parámentro sea un puntero, habrá que pasar una referencia a la estructura. Así, y suponiendo que se haya definido la estructura como `GsIMAGE graf`, la llamada a la función sería como sigue:

```
GsGetTimInfo(0xXXXXXXXX,
&graf);
```

Tras esto, en la estructura `Grafico` tendremos una útil información que nos ayudará a mover el gráfico y su *CLUT* a la posición correcta en el *Frame Buffer*. En *pmode* se ha almacenado el modo de *pixel*, que puede tomar los siguientes valores:

0 - CLUT de 4 *bits* (16 entradas)
1 - CLUT de 8 *bits* (256 entradas)
2 - Color Directo de 16 *bits*
3 - Color Directo de 24 *bits*
4 - Otros modos

La posición del gráfico y su CLUT dentro del *Frame Buffer* quedan almacenados en `px`, `py` y `cx`, `cy` respectivamente, y las dimensiones de ambos en `pw`, `ph` y `cw`, `ch`. Los punteros a los datos del gráfico y el CLUT quedan almacenados en `pixel` y `clut`. Ahora que tenemos la información necesaria para realizar el traslado de la *RAM* al *Frame Buffer*, ya sólo queda hacer dos llamadas a `LoadImage` para pasar en dos tandas el gráfico (o *bitmap*) y el *CLUT*. La cosa quedaría más o menos como sigue:

```
rp.x=graf.px; rp.y=graf.py;
rp.w=graf.pw; rp.h=graf.ph;
rc.x=graf.cx; rc.y=graf.cy;
rc.w=graf.cw; rc.h=graf.ch;
LoadImage(&rp, pixel);
LoadImage(&rc, clut);
```

`rp` y `rc` son dos estructuras `RECT` que se deben haber declarado con anterioridad. De nuevo hemos derreferenciado `rp` y `rc`, ya que *LoadImage* exige que le pasemos un puntero a la estructura `RECT`. Con lo hecho, hemos conseguido pasar el gráfico en formato *TIM* desde la memoria *RAM* al *Frame Buffer*. Hemos sentado las bases para que el mes que viene podamos trabajar con *sprites* sin ningún problema. Para ello contaremos con interesantes ejemplos.

Estructuras y Funciones

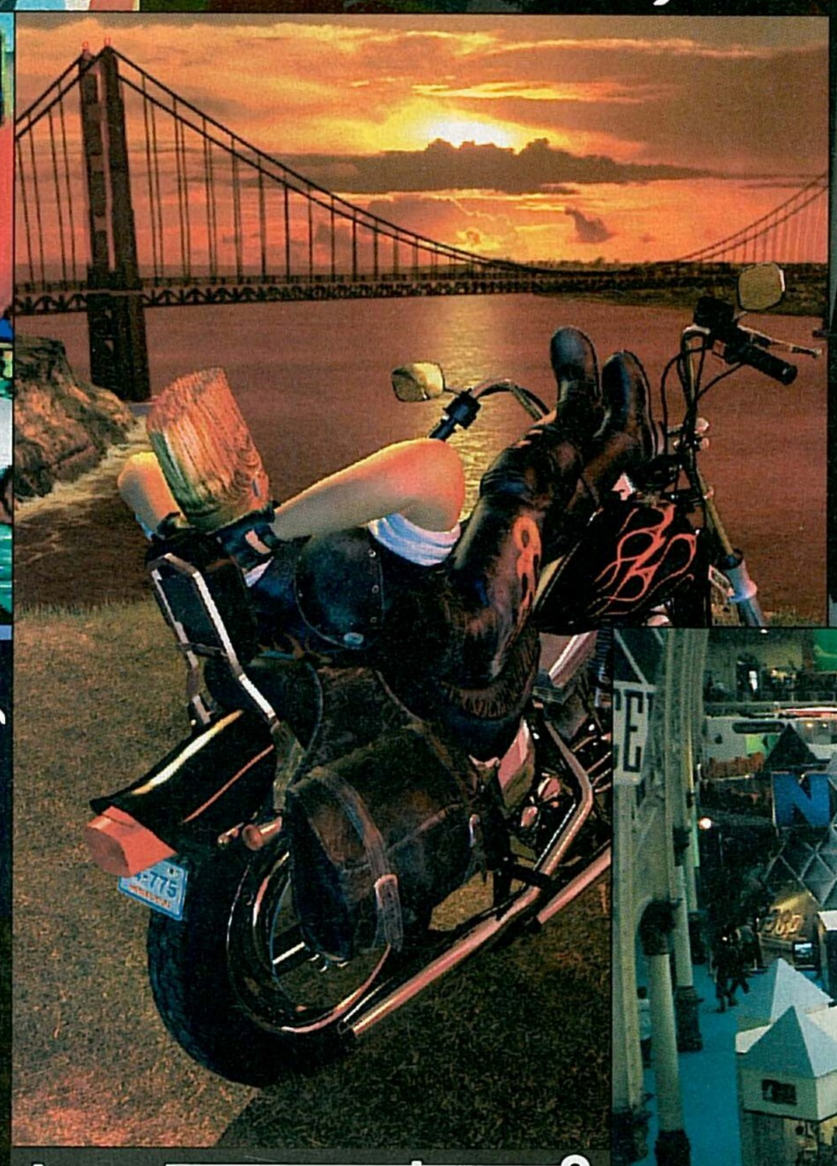
Este mes sólo hemos trabajado con las estructuras RECT y GsIMAGE y las funciones *GsGetTimInfo* y *LoadImage*. El mes que viene será necesario profundizar en el resto de estructuras y funciones.

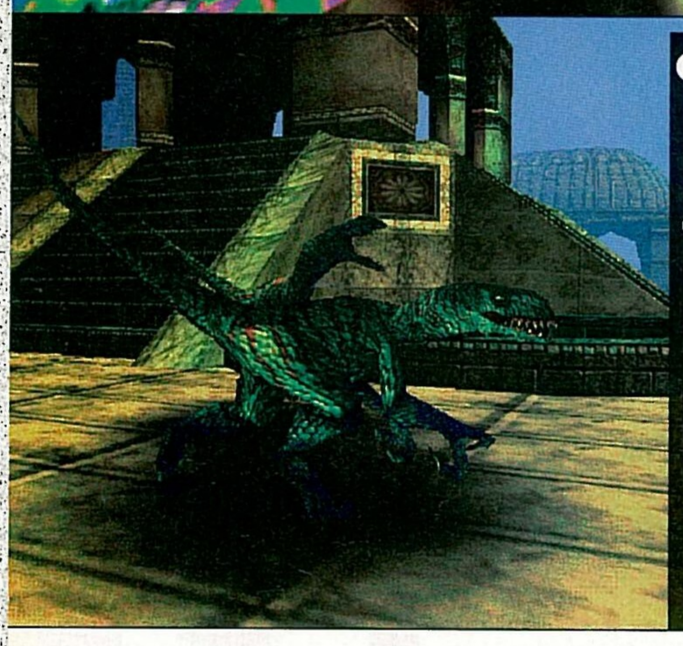
ESTRUCTURAS

```
typedef struct RECT {
    short x,y;
    short w,h; } RECT;

struct GsIMAGE {
    short pmode;
    short px,py;
    unsigned short pw,ph;
    unsigned long *pixel;
    short cx,cy;
    unsigned short cw,ch;
    unsigned long *clut; }

struct GsSPRITE {
    unsigned long
attribute;
    short x,y;
    unsigned short w,h;
    unsigned short tpage;
    unsigned char u,v;
    short cx,cy;
    unsigned char r,g,b;
    short mx,my;
    short scalex,scaley;
    long rotate; }
```

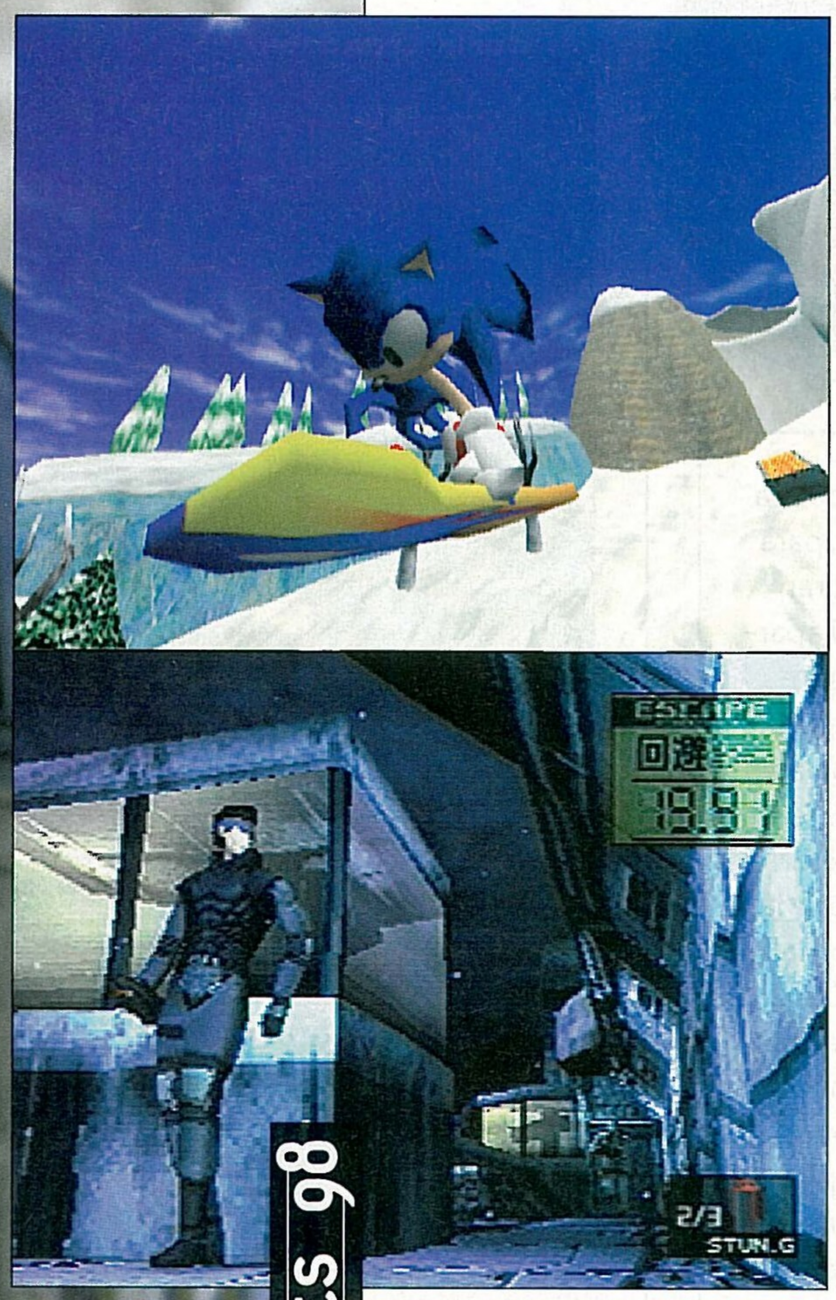
FUNCIONES

```
void GsGetTimInfo (
    unsigned long *tim,
    GsIMAGE *im )

int LoadImage (
    RECT *recp,
    u_long *p )

int StoreImage (
    RECT *recp,
    u_long *p )

int MoveImage (
    RECT *recp,
    u_long *p )

void GsSortSprite (
    GsSPRITE *sp,
    GsOT *otp,
    unsigned short pri )

void GsSortFastSprite (
    GsSPRITE *sp,
    GsOT *otp,
    unsigned short pri )
```

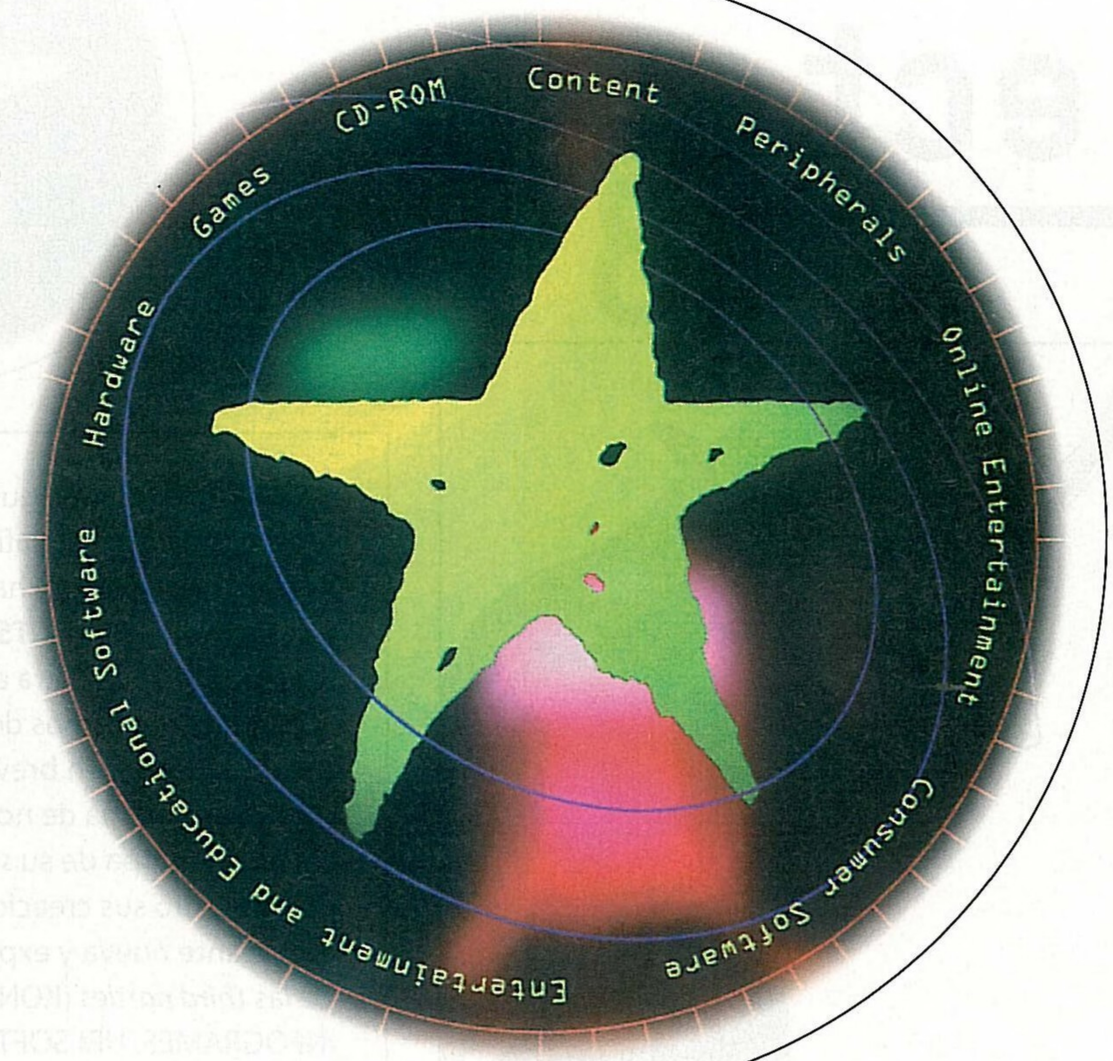




COMIENZA OTRA



*La feria londinense de Septiembre ha servido para confirmar lo que en **Atlanta** eras simples promesas, y para descubrir que la «guerra» de **N64** por desbancar a **PSX** del liderato mundial está servida.*

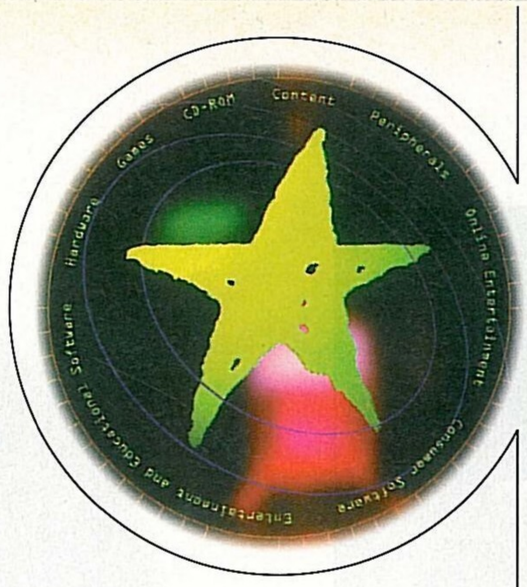


GUERRA

La penúltima gran cita internacional del mundo del videojuego en este año, el ECTS londinense, vino a confirmar con hechos lo que desde hace meses eran simples tendencias o meras promesas. Con *PLAYSTATION* consolidada como monarca absoluto del sector, era el momento de que *NINTENDO 64* ofreciera una demostración palpable, abundante y variada en juegos, del potencial de su futuro arsenal para el asalto al trono. Pues bien, así fue y pensamos que esa es la mejor noticia de la feria junto a la espectacular *demo* de SONIC ADVENTURE para *DREAMCAST*. Por otro lado, también comprobamos el ascenso imparable de compañías como INFOGRAMES, ACTIVISION o UBI SOFT, que darán que hablar en los próximos meses. Para finalizar, mención especial a KONAMI y su METAL GEAR SOLID para *PLAYSTATION*, el mejor juego de la feria a juicio de todos.

por DE LUCAR & A. DREAMER

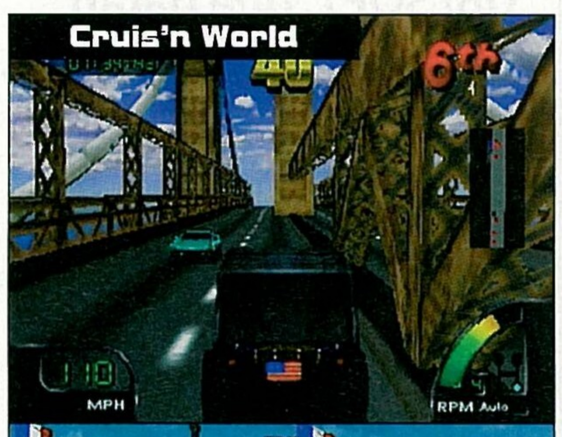
ects 98



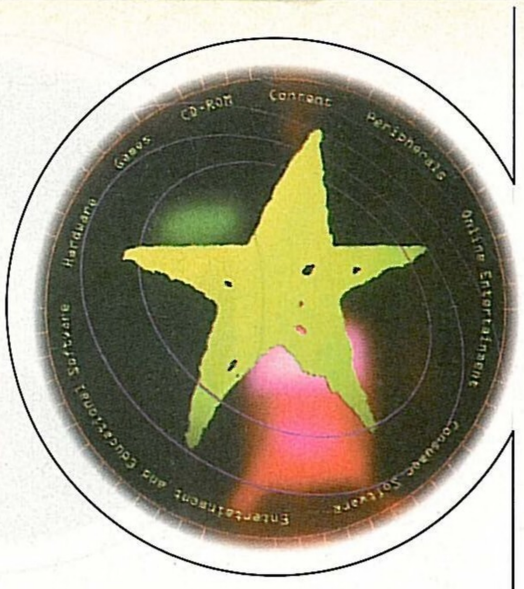
Aunque habrá alguien que pueda cuestionar esta afirmación, para nosotros ***NINTENDO 64*** ha sido la gran revelación de este **ECTS´98** en el apartado de consolas. Para alegría de todos, por fin podemos decir que este sistema contará en breve con una auténtica avalancha de novedades. Bajo la enorme carpa de su *stand*, NINTENDO presentó sus creaciones junto a la abundante nueva y explosiva hornada de las *third parties* (KONAMI, ACCLAIM, INFOGRAMES, UBI SOFT, etc) que, ahora más que nunca, parecen volcadas en programar para ***N64***. Entre todos los títulos destacaba con luz propia THE LEGEND OF ZELDA: OCARINA OF TIME, el centro de todas las miradas y el título que mayores colas formó para probarlo. ROGUE SQUADRON, la segunda entrega de STAR WARS para ***NINTENDO 64***, pese a estar aún en fase de programación presentaba un aspecto increíble y promete ser aún más espectacular que su antecesor. También hicieron las delicias de los presentes F-ZERO X, F1 WORLD GRAND PRIX, 1080º SNOWBOARDING, BANJO KAZOOIE, TUROK 2, ISS´98, WIPEOUT 64, V-RALLY´99, BOMBERMAN HERO, HOLY MAGIC CENTURY, WAIALAE GOLF y STARSHOT. Para finalizar con lo visto en este *stand*, NINTENDO mostró también la nueva ***GAME BOY COLOR*** y dio a conocer algunos de los futuros títulos específicos de la nueva portátil, aunque no pudimos ver aún ninguna muestra.

Todos estáis familiarizados con estos tres títulos para NINTENDO 64. Dada su gran calidad y su inminente salida al mercado (F1 WORLD GRAND PRIX ya está en las tiendas), no podían faltar en tan importante cita.

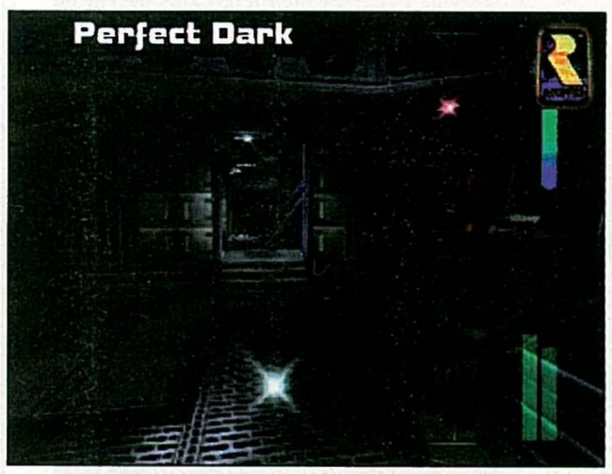
NINTENDO



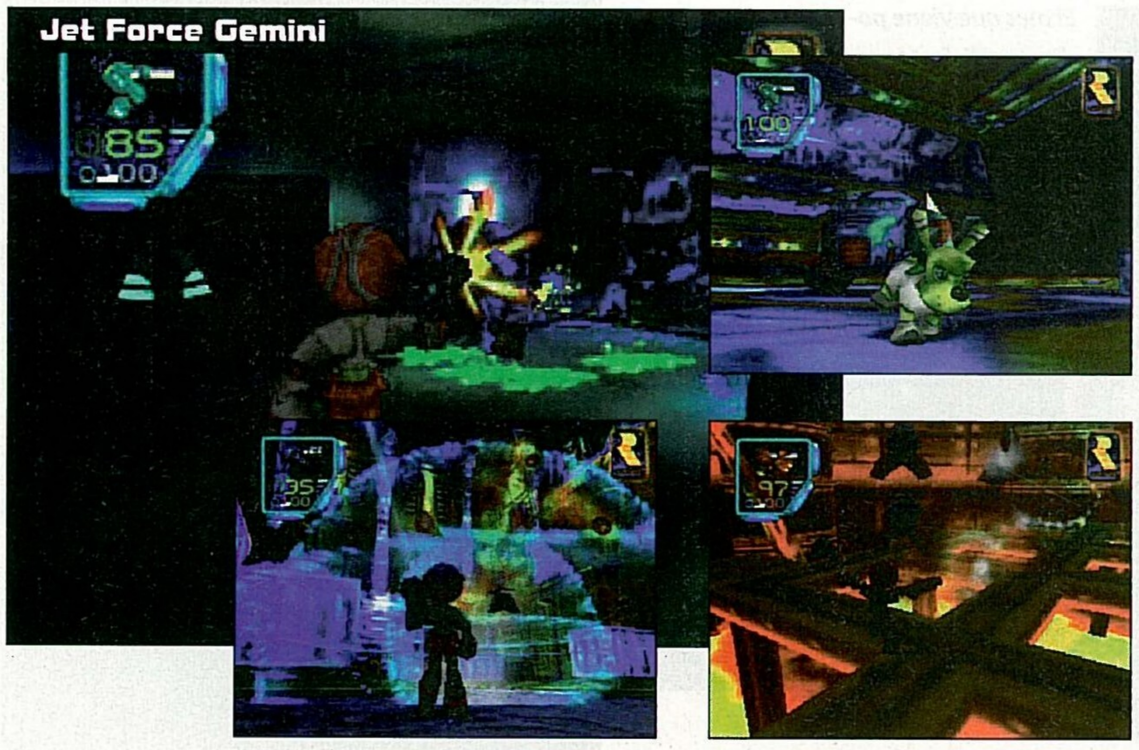
ects 98



Este plataformas 3D de RARE nos trae al recuerdo un título de **SUPER NINTENDO**, *MR. NUTZ. Aunque las diferencias son notables por el cambio de plataforma y la dinámica de juego, la belleza de sus gráficos y sus protagonistas hacen que la referencia sea inevitable.*



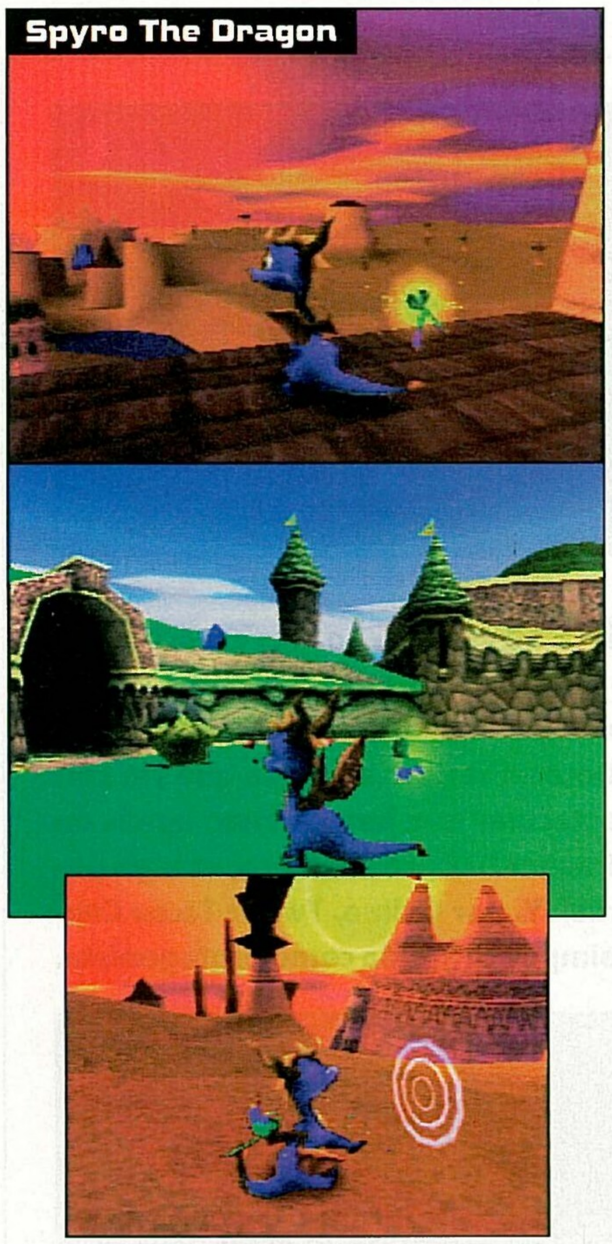
La nueva hornada de RARE. Lo más espectacular del *stand* de NINTENDO fue el impresionante vídeo con tres de los futuros títulos de RARE. Cualquiera de los tres merecería encabezar este recuadro especial, pero PERFECT DARK, lo que sería la continuación de GOLDENEYE pero sin nada que ver con 007, es el *shoot´em-up* más explosivo y espectacular que hemos visto jamás en consola. JET FORCE GEMINI, que no es inferior a PERFECT DARK en absoluto, es algo parecido pero desde un planteamiento menos real. Y, por último, TWELVE TALES: CONKER 64 es un alucinante MARIO 64 pero con una simpática ardilla como protagonista.



RARE ha combinado toda la acción de un shoot'em-up en 3D con puzzles, obstáculos y hordas de enemigos en un impresionante título para **NINTENDO 64**.

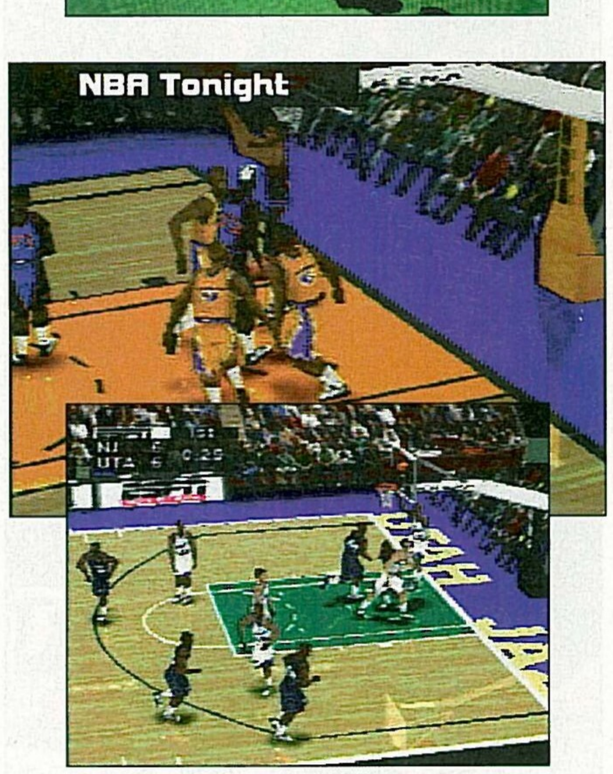
RARE ha realizado su primera incursión en el mundo de **GAME BOY COLOR** *con una versión de su juego para NINTENDO* CONKER´S TALES. *Un joya para tu portátil.*

ects 98



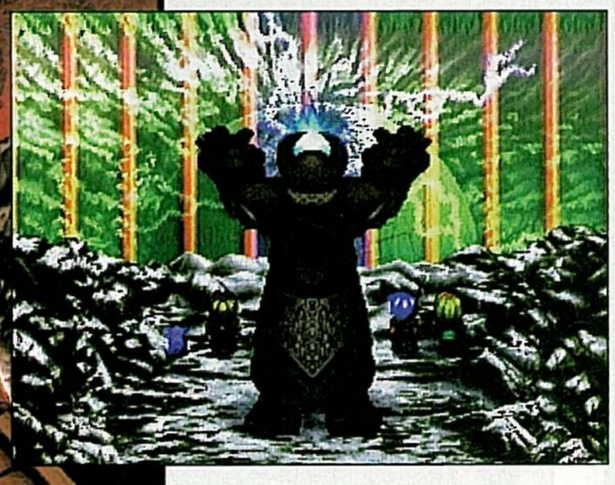
SPYRO es, sin lugar a dudas, uno de los mejores exponentes de la nueva generación de juegos para PLAYSTATION de SONY.

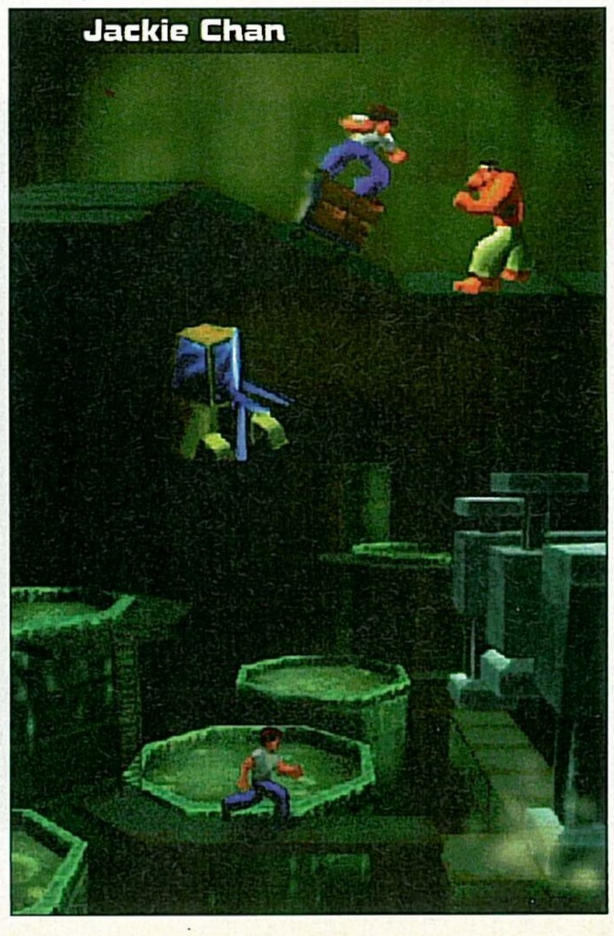
SONY, como es habitual en todas las ferias, acaparó buena parte del interés de los asistentes con un espectacular y vanguardista *stand* absolutamente repleto de expositores con juegos. En esta instalación, una de las más grandes de la feria, mostró los que serán sus futuros lanzamientos para esta campaña navideña y buena parte de los títulos que abrirán la próxima campaña. TEKKEN 3, como se esperaba, fue la estrella del *stand,* pero también destacaron la presencia de juegos como SPYRO, JACKIE CHAN, CRASH BANDICOOT 3, BUG´S LIFE, LIBERO GRANDE, POINT BLANK 2, NBA TONIGHT y BUST A GROOVE. Como veis, alguna sorpresa aunque pocas novedades. Es lógico por la tremenda labor de promoción que ***PLAYSTATION*** suele hacer con todos los títulos y que nos permite informaros de cada título con varios meses de antelación. Alguno pensará que SONY no guarda nada para las ferias, pero no es así. Por ejemplo, allí pudimos entrevistarnos con algunos de los máximos reponsables de la compañía, como **Chris Deering** y **Juan Montes**, y comprobar la evolución de los programadores de ***NET YAROZE*** con un español, **Jesús Ventoso**, entre los más destacados con un «TEKKEN» con unas maneras excelentes. Por otro lado, conocimos más datos de la página WEB de ***PSX*** y asistimos a la fiesta de SONY, la más espectacular que han visto nuestros cansados ojos en 30 años.



SONY C.E.

El mes que viene podremos disfrutar con las aventuras y las increíbles melodías de WILD ARMS.





ects 98

*Una de las estrellas indiscutibles del **ECTS**, la superproducción de NAMCO, a pesar de ser de sobra conocida por todos nosotros, sigue sorprendiéndonos.*

ENTREVISTA

CHRIS DEERING ■ presidente s.c.e. europa

JUAN MONTES ■ director de desarrollo

P. *Sr. **Dearing**, ¿podría hacernos un breve comentario sobre la marcha actual de **PLAYSTATION** en el mercado europeo?*

R. Bueno, actualmente en **Europa** llevamos 10 millones de **PLAYSTATION** vendidas y, según nuestras previsiones, en el 99 esperamos alcanzar los 15 millones. Para lograrlo deberemos seguir ofreciendo juegos que aprovechen al máximo el *hardware* de **PSX**, como GRAN TURISMO o RESIDENT EVIL 2. En cualquier caso, las apuestas más fuertes en *software* llegarán de la mano de los programadores de SONY y NAMCO.

P. *Ahora que SEGA va a lanzar su nueva consola, **DREAMCAST**, ¿es posible que SONY nos sorprenda con **PLAYSTATION 2**?*

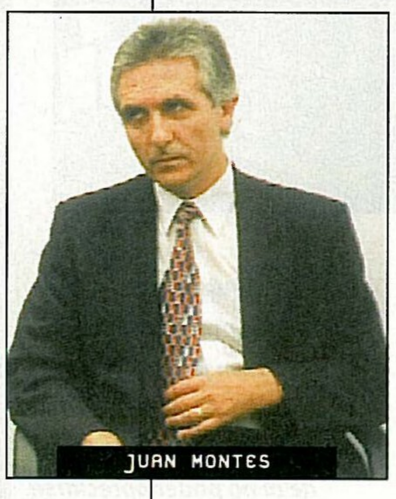
R. (C.D.) Aprovecharemos el tirón que tiene actualmente la consola hasta el año 2000 explotando las posibilidades de las nuevas configuraciones de *pads* como el *Dual Shock*. Nosotros no vamos a cambiar nuestros planes por la aparición de **DREAMCAST** porque tenemos un amplio mercado establecido.

CHRIS DEERING

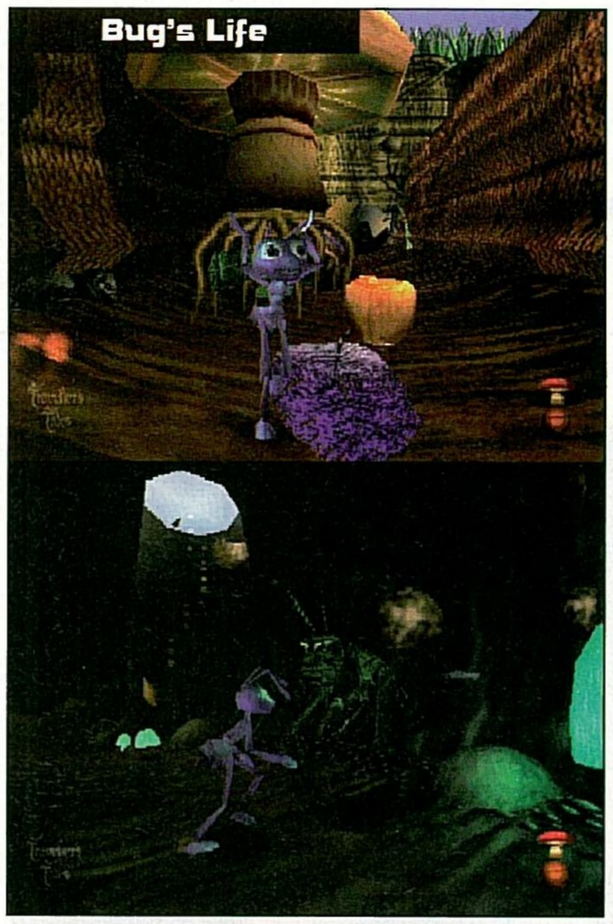
P. *¿Cree que ZELDA 64 puede afectar a las ventas de **PSX**?*

R. (J. M.) No, definitivamente no. Hoy en día, hasta el **PC** es un competidor y nosotros estamos arriba.

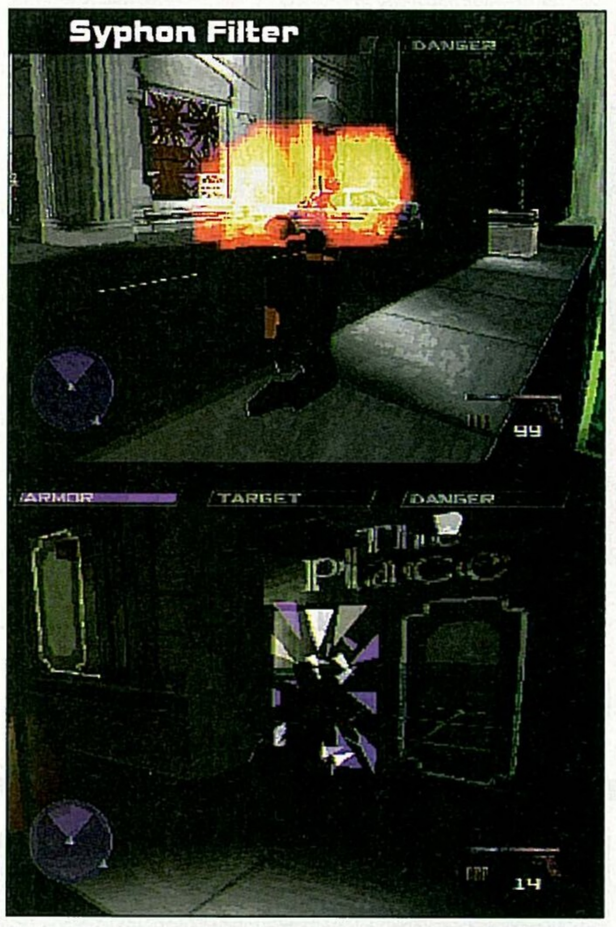
P. *¿Cuál cree que es el secreto del éxito de **PLAYSTATION**?*


JUAN MONTES

R. (J.M.) El formato CD, además de abaratar los costes con respecto a los cartuchos, permite una mayor calidad en el sonido de los juegos, sobre todo en la música. También ha influido la tecnología punta que incorporaba en el momento de su lanzamiento, que aún nos permite realizar cosas realmente increíbles y, por su puesto, la enorme calidad de los juegos que han salido.

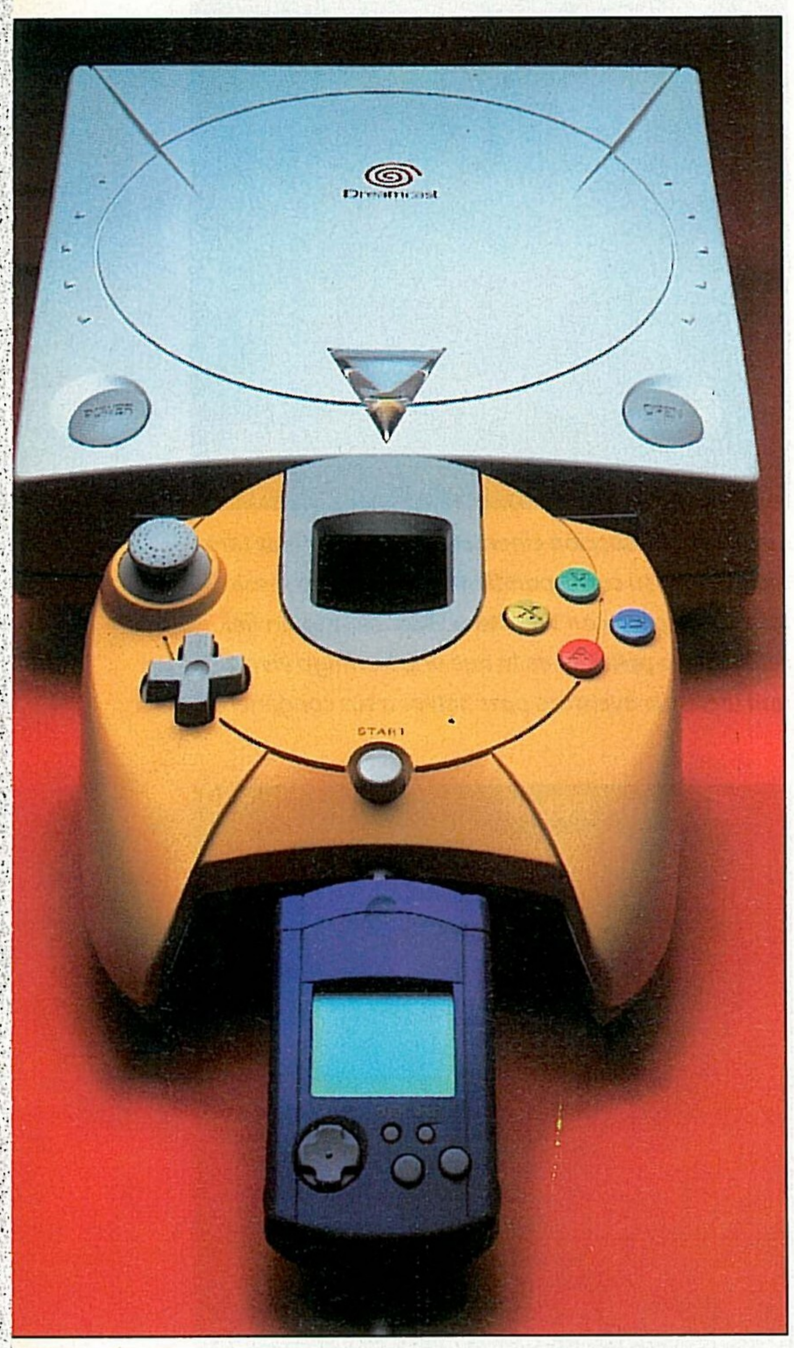


*La última producción cinematográfica de **Pixar** también tendrá su correspondiente versión en consola, tal y como pasó con **Toy Story**. **BUG´S LIFE** es un fiel reflejo de la película, en la que una hormiga vivirá una increíble aventura para salvar a sus congéneres.*



*Desde los estudios norteamericanos de SONY nos llega **SYPHON FILTER**. Una aventura de espionaje en la línea de Metal Gear, con dos personajes jugables, 16 niveles, 4 jefes finales y un completo arsenal para evitar que unos terroristas contaminen la atmósfera.*

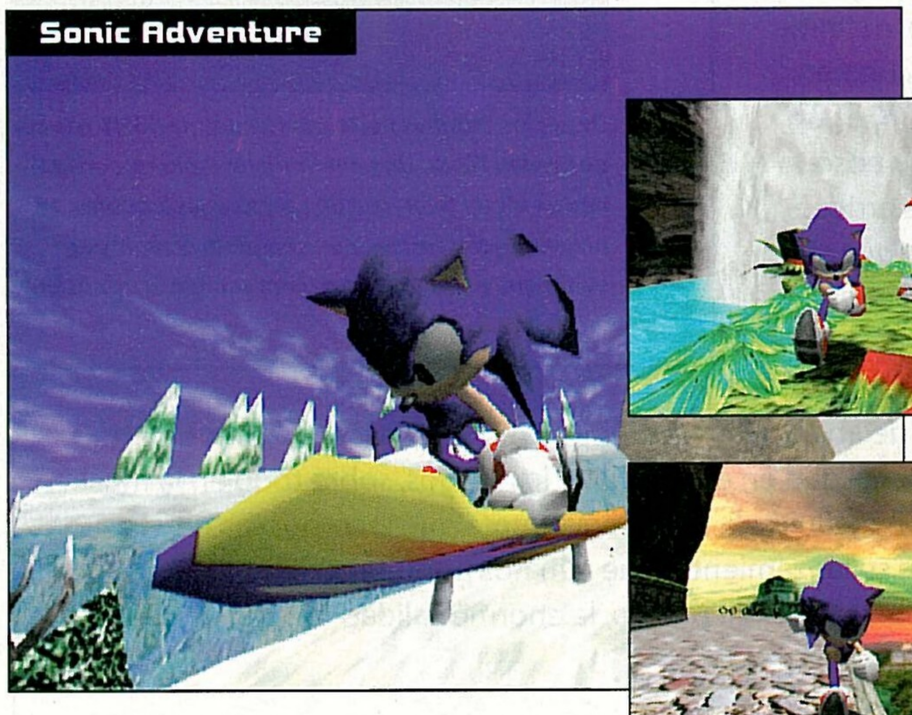
ects 98

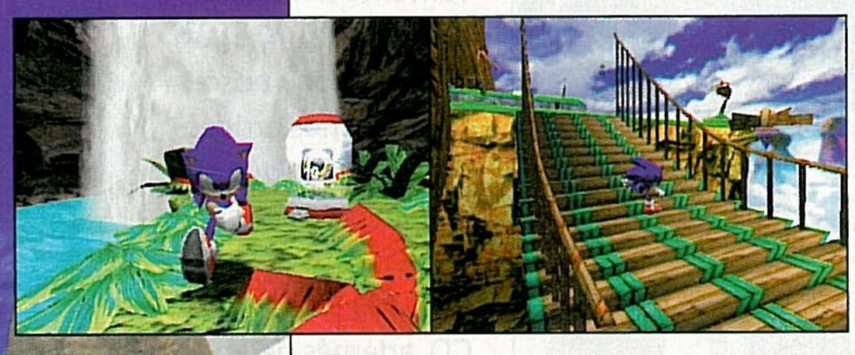


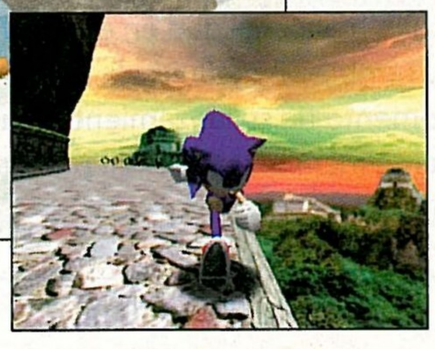
Alejada del bullicio de la feria, SEGA decidió establecer sus cuarteles en los salones del Hotel Hilton para la presentación de **SONIC ADVENTURE** para ***DREAMCAST***. **José Angel Sánchez**, Consejero Delegado y Director General de SEGA **España**, fue el encargado de hacer la presentación de la consola y el vídeo en el que **Shoishiro Irimajiri**, Presidente de SEGA, explicó las líneas maestras de esta nueva etapa de la compañía. El punto cumbre de la rueda de prensa llegó con la posterior exhibición de una impresionante *rolling demo,* instalada en la misma consola, de SONIC ADVENTURE. Resultaría complicado describir con fidelidad lo que vimos en esos escasos diez minutos de juego, pero se podría resumir en que es lo más espectacular que ha contemplado nuestra retina. Con una calidad gráfica y técnica sin precedentes, y que sólo podría compararse a los niveles de preciosismo de la mejor película de animación por ordenador, SONIC ADVENTURE nos invita a revivir el estilo de juego clásico de la saga pero bajo una mpresionante perspectiva en 3D. El resultado es asombroso.

***En SONIC ADVENTURE podremos jugar con cualquiera de los tradicionales personajes de la saga, a excepción, claro está, del malvado* Doctor Robotnik.**

S E G A







Aunque parezca increíble, las pantallas que veis no hacen justicia a la realidad. Gran parte de su espectacularidad se pierde al no poder apreciarse la inigualable calidad de sus movimientos.